Robotics Platforms: A Research Review of an Open-Source Robotics and Digital Twin Development Repository

Author:

Paul George Savluc
CEO & Founder, OpenQQuantify, Tomorrow's AI, Robotics Platforms and The Medical Technology

Co-Authors and Company Members:

Robotics Platforms Engineering Team: Muhammad Hamza Shahzad, Afsana Afroze, Shrideshi Samaraweera, Soumyadeep Das, I.K., Mahesh Pun, Daniel Moussa, Ezekiel Ochuko, Tchinda Jordan Peggy, Neha Medehal Reddy, Valencia Walker, Paul Karanja,

The Medical Technology Team: Bishal Kharel, Vasu Krishna

Not all company members are mentioned. If you want to be featured in our papers please reach out to Paul George Savluc.

Introduction

Digital twins have become a pivotal technology for both hardware engineering and robotics. A 2024 McKinsey explainer describes a digital twin as a **virtual replica** of a physical object, person or process that is linked to real data sources so that the model updates in real time. By simulating the behaviour of the real world in a digital environment, digital twins let engineers test scenarios safely and make better decisions. In robotics, digital twins provide a safe rehearsal space for offline programming and real-time monitoring, allowing developers to visualise and test scenarios before applying them to physical robots. Open-source communities are accelerating the adoption of digital-twin technology; the Digital Twin Consortium's open-source collaboration initiative invites consortium and non-consortium members to co-develop models, tools and standards on GitHub.

Robotics Platforms is an open-source project on GitHub that leverages artificial intelligence, digital twins and immersive 3-D environments to simplify electronics design and robotic system prototyping. It combines a FastAPI/Flask backend, a CesiumJS-based front end and an optional AI layer to provide real-time simulations, remote control and hardware design assistance. This research paper documents the repository's architecture and code, examines its capabilities and highlights opportunities for community participation.

The live link for the fully hackable Robotics and Digital Twin IDE can be found here: https://www.roboticsplatforms.com/

You can check daily for new apps and features, and if you would like to sponsor or become a partner or client the team is always happy to help.

Repository Structure and Core Capabilities

The `Robotics-Platforms` repository is organised into several top-level directories (Figure 1) that collectively enable digital-twin simulation, AI-assisted design, 3-D asset management and hardware-specific services.

- **`backend/`** – A Python package built with FastAPI that serves as the REST API for the digital-twin platform. It exposes endpoints for health checks, geolocation, public data queries, telecommunications, simulations and simple AI tasks. The backend also contains simulation state classes and services for geolocation and satellite orbit calculations.

- **`frontend/`** – A JavaScript/HTML front end based on CesiumJS. It provides a 3-D globe view, user interface controls (search, start/stop simulation, object placement, AI chat), physics-based simulation loops and Playwright end-to-end tests.

- **`ai_integration/`** – AI integration code (currently a stub) that shows how to connect large language models (LLMs) to interpret user queries. An `action_schema.json` file defines actions such as `BRAKE`, `STOP` and `MOVE_TO`, and `ai_integration.py` demonstrates using OpenAI's API to process text.

- **`physics/`** – A set of JavaScript modules implementing simple physics behaviours like aircraft thrust, buoyancy and collision detection for the simulation.

- **`3d_objects/`** – A placeholder directory for user-uploaded 3-D models (.glb, .gltf, .obj, .fbx). A JSON registry maps object names to physics attributes and model paths.

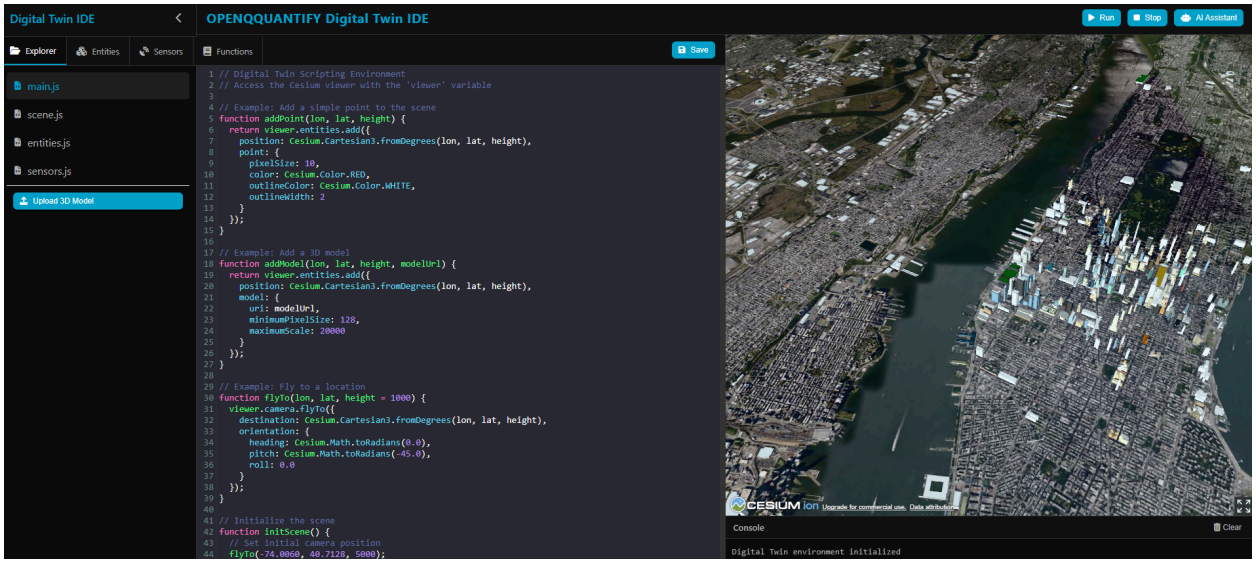
- **`templates/`** – Jinja2/HTML templates that render the digital twin UI. The main template (`digital_twin.modular.html`) loads CesiumJS, defines UI elements (search, simulation controls, AI panel) and imports modules for scene initialisation and simulation management.

- **`docs/`** – Project documentation describing the system architecture and simulation flow. For example, the **architecture** document states that the front end uses CesiumJS, the back end uses FastAPI/Flask and the AI layer is modular. A simulation flow document explains that a `SimulationManager` binds entity update methods to viewer events, runs a physics loop on each pre-update and triggers AI classification when objects are placed.

- **`scripts/`** – Utility scripts such as `download_models.py` for downloading remote 3-D models into the assets folder.

Figure 1 – Digital Twin User Interface

The front-end interface integrates CesiumJS and features controls for search, simulation start/stop, object placement, loading buildings, toggling labels or satellite imagery and opening an AI chat panel. A still image from the repository's digital-twin demonstration is shown in Figure 1.



Core Capabilities

The README highlights five core capabilities that reflect the project's design goals:

1. **AI-powered design** – Utilises structured language models and generative AI to automate circuit design, produce schematics, optimise field-programmable gate array (FPGA) layouts and analyse power management.

2. **Digital twins** – Provides real-time virtual replicas for performance optimisation, predictive analytics, failure scenario simulation and remote monitoring. Public API endpoints supply normalized coordinates (`/api/geo/area`), OpenStreetMap data (`/api/public-data`) and simulation results (`/api/simulation/zone`).

3. **Embedded systems** – Offers firmware development expertise and bespoke hardware solutions.

4. **AI integration services** – Supports custom AI agent development, process automation and natural-language interfaces.

5. **Immersive engine** – Delivers a web-based 3-D environment with physics-based rendering, WebGL optimisation and multiplayer support.

These capabilities align with the wider digital-twin landscape. McKinsey emphasises that digital twins link virtual replicas to real data sources, enabling simulations, scenario planning and improved decision-making. RoboDK's analysis notes that digital twins provide a safe environment to visualise and test robotic programming, delivering precision and safety benefits.

Back-End Architecture

FastAPI Server

The primary entry point for the back end is `backend/src/server.py`. It performs the following tasks:

- **Static and template configuration:** The server loads environment variables (e.g., a Cesium Ion token) and mounts static files from the `frontend/static` folder. A Jinja2 `TemplateResponse` renders the digital twin template and injects the token when allowed.

- **Root and health routes:** The root endpoint returns basic service information and links to the UI and API docs. A `/health` endpoint returns `{"status":"ok"}` for monitoring.

- **API routers:** The server imports routers for geolocation, public data, telecom and simulation, and registers them with the FastAPI app. Each router is defined in `backend/src/routes`.

- **AI endpoints:** A `/ai_query` endpoint demonstrates how to parse user messages; it geocodes "drive to" commands via Nominatim, extracts coordinates from free-form text, or returns a fallback message when AI is not configured. A `/ai/object` endpoint classifies a 3-D model (e.g., car, aircraft, satellite) and returns suggested actions.

API Routes and Services

- **Geolocation router** (`geo_routes.py`) – Implements a simple `/api/geo/info` endpoint that returns the latitude and longitude provided by the client.

- **Health router** – Provides a `/health` endpoint for service status.

- **Public data router** – Returns nearby OpenStreetMap elements around a given coordinate and optionally synthesises satellite positions. The `osm_services.py` service builds an Overpass API query and handles network errors gracefully by returning an empty list.

- **Telecom router** – Returns a simple status response; it is currently a placeholder for telecom integrations.

- **Simulation state classes** – `EntityState` stores an entity's id, type, position, velocity and status. `SimulationState` maintains a dictionary of entities and updates their positions each tick based on velocity.

- **Services** – `geolocation_services.py` provides coordinate normalisation and fetches OSM objects. `environment_loader.py` extracts speed zones and police checkpoints from OSM tags. `satellite_orbit_service.py` computes simple orbits for synthetic satellites.

Front-End Architecture and Simulation Flow

The Cesium-based front end resides under `frontend/` and `templates/`. The modular template `digital_twin.modular.html` includes script tags for CesiumJS, CodeMirror (for code editing) and custom modules. It defines a user interface with search boxes, simulation controls, model selection, object placement buttons and an AI panel. The script imports functions such as `initScene`, `createSimulation` and `enableClickToPlace` from `/static/js` and attaches event handlers to UI buttons. When the user clicks **Start**, a `SimulationManager` is created and its `update` method is bound to the Cesium viewer's pre-update events.

The `docs/simulation_flow.md` document describes the simulation lifecycle:

1. **Initialization:** `initScene()` creates a Cesium viewer, chooses imagery/terrain based on Cesium Ion availability and starts the physics loop.

2. **Simulation managers:** Entities such as vehicles and aircraft expose `update` methods for acceleration, braking and waypoint setting.

3. **Physics loop:** A central `physicsStep(dt)` applies rigid body integration, buoyancy and collision handling each pre-update.

4. **Event triggers:** Checkpoints and triggers (e.g., police stops) are evaluated each update to influence entity behaviour.

5. **Object placement & AI:** Clicking the globe allows users to place 3-D models; the front end posts to `/ai/object` for classification and suggestions.

6. **Persistence:** Entities and user selections are kept in memory; persistent storage is a planned enhancement.

The **physics** directory contains simple JavaScript functions used by the simulation manager. For example, `applyAircraftPhysics` adds thrust to an aircraft's vertical velocity, `applyBuoyancy` applies upward force when an entity is below a water level and `handleCollisions` resolves collisions by reversing vertical velocities when objects come within a threshold distance. These functions illustrate how the simulation currently focuses on proof-of-concept physics rather than comprehensive dynamics.

AI Integration

The repository's AI layer is currently a minimal demonstration. The `ai_integration/ai_integration.py` file defines an `AIEngine` class that loads an API key from the environment and sends user queries to OpenAI's API. The response text is returned as the AI result. A separate `action_schema.json` defines possible actions (e.g., `BRAKE`, `STOP`, `MOVE_TO`) and their targets. In its present state, AI responses are not integrated into the simulation; the `/ai_query` route in the backend provides fallback logic that recognises "drive to" commands via geocoding and extracts latitude/longitude pairs from user messages. Further development could expand the AI layer to interpret natural-language instructions into structured simulation commands or generate code to control hardware.

3-D Models and Sensor Configurations

A key feature of the digital-twin platform is the ability to display and interact with 3-D models. The `3d_objects` directory is a placeholder for user-uploaded models; an `objects_registry.json` file associates object names with model paths and physics types. The `scripts/download_models.py` script downloads models from provided URLs and stores them in the assets folder. This design lets contributors easily expand the library of vehicles, aircraft and sensor devices.

The `sensors_pheripherals_radios` folder contains a `sensor_parameter_configurations.js` file that defines configurable parameters for ultrasonic, radar, LiDAR, proximity, temperature, humidity and pressure sensors. Each sensor

has attributes like range, field-of-view and colour. These parameters can be used by the front end to visualise sensing cones or spheres on the globe. Contributors can add new sensors or refine the existing configurations.

Testing and Continuous Integration

The repository incorporates testing at multiple levels:

- **Backend tests:** Pytest tests for health and public data endpoints reside in `backend/tests` and can be executed with `python -m pytest backend/tests`. A helper script `check_endpoints.py` uses FastAPI's `TestClient` to verify that endpoints respond correctly.

- **Front-end tests:** Playwright end-to-end tests in `frontend/tests/e2e` validate UI functionality such as search autocomplete, object placement and building loading. Running `npm run test:e2e` in the `frontend` directory installs Playwright and runs the tests.

- **CI pipeline:** A GitHub Actions workflow installs dependencies, runs tests and executes `pip-audit` for dependency vulnerability checks. This ensures that new pull requests maintain code quality and security.

What the Digital Twin IDE already is (high level architecture)

Front end: `digital_twin.html`

You have a single page app that embeds:

- A Cesium `Viewer` with world terrain enabled and a minimal UI surface (no timeline/animation widgets).
  digital_twin
- An inline "multi file" code editor concept (`main.js`, `scene.js`, plus dynamic "model config" buffers) backed by `state.files`.
  digital_twin
- A "Run" pipeline that:
  - clears scene entities and event handlers,
  - sanitizes user code,

    - executes the code against `(viewer, Cesium, console)` using `new Function(...)`.
      digital_twin
- Entity UX:
    - draggable entities via `ScreenSpaceEventHandler`
    - tooltip on hover
    - a properties panel to edit name and position
    - update/delete flows
- A sensors palette with drag and drop that creates simplified sensor visuals (cone via cylinder, sphere via ellipsoid, point).
- A built in AI Assistant panel that sends your **full IDE context** to the backend at `/ai_query` and applies returned diffs as patches.
  digital_twin

Backend: `Server_Host.py`

Your Flask server is already doing something unusually valuable: **AI assisted refactors safely**.

- Serves the IDE template at `/` and injects `CESIUM_ION_TOKEN` into the HTML.
  Server_Host
- Provides `/ai_query` which:
    - requires `ide.files` to be present (multi file context is mandatory),
    - builds a deterministic system prompt including an index of function names per file,
    - forces the model to output JSON patches only,
    - applies the diffs server side in a dry run,
    - blocks forbidden JS patterns and warns on “physics risk” patterns before returning patches.

That “patch + dry run + guard rails” design is the right skeleton for a serious IDE.

Front end walkthrough (what each subsystem is doing)

Cesium viewer initialization

Your viewer configuration:

- Uses `Cesium.Terrain.fromWorldTerrain()`
- Disables a bunch of default UI widgets
- Enables shadows
- Then `initializeScene()` enables lighting, depth test against terrain, fog, HDR, and loads OSM buildings asynchronously.

Important detail: you also do `viewer.scene.globe.depthTestAgainstTerrain = true;` which is great for preventing underground artifacts, but it also changes picking behavior and can surprise entity placement logic if you rely on `pickPosition` vs globe picking.

Multi file editor model

You've got a very workable concept:

- `state.currentFile` and `state.files` hold "buffers"
- `loadFileContent(filename)` saves the current editor buffer into `state.files[current]` then loads the requested file, using fallback templates for `main.js` and `scene.js`.

Right now, the "Save" button does not actually persist. It only logs "Code saved".

digital_twin

So the IDE feels like an IDE, but behaves like a session toy unless you add persistence.

Run pipeline (execution + sanitization)

This is the heart:

```
const safeCode = sanitizeCode(code);

const func = new Function('viewer', 'Cesium', 'console', safeCode);

func(viewer, Cesium, customConsole);
```

And `sanitizeCode()` blocks patterns like `eval`, `new Function`, `document.`, `window.`, `localStorage`, `fetch`, etc.

Two key observations for improving the Digital Twin:

1. Banning `new Function(` but still using `new Function` to execute it. That's normal for a quick IDE, but it is still an escape surface if someone finds a Cesium or JS runtime gadget.
2. Ban `fetch(` in user code, which is good, but users can still hang the browser with heavy computation unless execution is isolated.

Entities (selection, dragging, properties)

- Dragging is implemented by picking the entity on mouse down, disabling camera rotate, then using `viewer.scene.globe.pick(ray, scene)` during mouse move.

- Tooltip and properties panel use `viewer.scene.pick` on hover and click.
  digital_twin
- The Properties panel updates entity coordinates and also repairs the `state.uploadedModels[name].entity` pointer if the name matches.
  digital_twin

Improvement: if you rename an entity, your `state.uploadedModels` dictionary is keyed by the old model name. You update the entity pointer if the new name exists as a key, but you do not rekey the dictionary. That can desync.

3D model upload (local + optional server persistence)

Your upload flow is actually two separate flows:

1. Always: create a `Blob URL` via `URL.createObjectURL(file)` and add it as `model.uri` so it renders for this browser session.
   digital_twin
2. Optionally: if “persist on server” is checked, POST `FormData` to `/upload-model`.
   digital_twin

Important: in the provided `Server_Host.py`, there is no `/upload-model` route. That likely lives in your registered blueprint `routes.objects`, but it is not in the snippet we have.

Server_Host

So today the checkbox may fail unless that blueprint implements it.

Sensors as first class scene objects

This is a nice early abstraction:

- You define `sensorParameters` with ranges, FOV constraints, descriptions, and display color.
  digital_twin
- Dragging a sensor onto the globe calls `createSensorEntity(sensorParam, earthPosition)` which renders a cone (cylinder), sphere (ellipsoid), or point.

Right now this is visualization, not simulation. It does not do ray casting, occlusion, returns, noise, or measurement streams.

AI assistant and patch application

This is the most “platform” part of the IDE:

- `collectIDEContext()` packages:
    - active file name,
    - a full copy of `state.files`,
    - a lightweight scene snapshot (count + up to 20 entity descriptors).
      digital_twin
- `processAIQuery()` sends `{ user_prompt, ide: collectIDEContext() }` to `/ai_query`.
  digital_twin
- Response patches are applied via a simple unified diff applier and then injected into `state.files` and editor if active.
  digital_twin

This is excellent, because it sets you up for:

- “AI writes code in your IDE”
- “AI refactors safely”
- “AI suggests changes without raw file replacement”

Backend walkthrough (what matters and why it’s good)

The AI patch contract (and why it’s the right design)

`Server_Host.py` forces:

- output must be JSON only
- changes must be unified diffs per file
- server dry runs the diffs
- forbidden patterns are blocked
- “physics risk” patterns generate warnings

This is basically “lint + policy + patch discipline” for an AI coding agent, which is the correct way to prevent your IDE from becoming a self modifying chaos machine.

Guard rails you already have

- Forbidden JS patterns: `eval`, `Function(...)`, `window`, `document`, `fetch`, `WebSocket`, `localStorage`, etc.
- Simulation risk heuristics: infinite loops, empty for loops, ultra fast timers, clock multiplier cap.

One upgrade: these regex checks are good, but a determined attacker can sometimes obfuscate around regex. Long term you want AST based validation for user code, and sandboxing for execution.

Community Participation

Contribution Guidelines

The README provides a clear procedure for opening a pull request:

1. **Create a feature branch:** Fetch the latest `main` branch, create a new branch and name it according to the change (e.g., `feature/docs-tests-pr`).

2. **Develop and test locally:** Group changes logically, run backend tests (`pytest backend/tests`) and front-end tests. Commit with descriptive messages.

3. **Push and open a PR:** Push the branch and open a pull request on GitHub. Include a summary of changes, test results and any security considerations.

The repository uses semantic commit messages (e.g., `docs: add testing and endpoints`) and expects contributors to avoid committing secrets (API keys). The `.env` file is ignored; an example `.env.example` is provided for configuration, and developers should use GitHub Secrets or Vault for production secrets. The project's `SECURITY.md` (not reproduced here) likely outlines responsible disclosure practices.

Areas for Contribution

The repository is still evolving; many components are rudimentary or marked as placeholders. Potential contribution areas include:

- **Expanding the AI layer:** Implement natural-language parsing to map user commands to simulation actions, integrate code generation for hardware control or design tasks and add support for multi-agent coordination. The `ai_integration` directory and AI endpoints provide starting points.

- **Improving physics models:** Enhance the physics modules to include kinematic constraints, vehicle dynamics, aerodynamics and environmental factors. Contributors could port physics engines like Ammo.js or integrate physics frameworks (Bullet, ODE) to improve realism.

- **Enhancing 3-D asset management:** Develop the Flask upload route mentioned in the `3d_objects` folder, store metadata in a database and allow collaborative model sharing. Extend the `objects_registry.json` to cover more robots, drones and sensors.

- **Sensor visualisation and radio modules:** Build interactive sensor modules that display sensing cones and volumes on the globe using the parameters defined in `sensor_parameter_configurations.js`. Provide functions to simulate sensor readings and integrate them with the AI layer.

- **Simulation persistence and multi-user support:** Add server-side state management and websockets to persist entities between sessions and allow multiple users to interact with the same simulation.

- **Documentation and tutorials:** The `docs/` folder could include tutorials on how to run the platform, integrate new sensors or create custom digital twin scenes. Providing example notebooks or videos would lower the barrier for new users.

Open-Source Ecosystem

Digital twin development benefits from community collaboration. The Digital Twin Consortium emphasises that open-source projects accelerate adoption and facilitate interoperability; consortium members and non-members can collaborate on code and collateral to bring digital twin technologies into the marketplace. By sharing models, simulation frameworks and data standards, developers can build reusable components for diverse domains, from robotics to energy systems. Robotics Platforms aligns with this philosophy by adopting the Apache-2.0 license, exposing public API endpoints and encouraging pull requests.

Roadmap to a Comprehensive Simulation & Digital Twin Platform for the Development of Robotics Platforms

Repository Overview

The **Robotics-Platforms** project is an AI-driven hardware engineering suite with initial support for “real-time simulation” of physical systems. Its codebase includes a Python backend exposing simple REST APIs, and a JavaScript/CesiumJS-based front-end for 3D visualization. The README touts “real-time digital twins” and sensor/actuator interfaces as core capabilities. Recent commits emphasize Cesium integration, physics simulation, and AI navigation support. In practice, the repository already provides basic geo-OSM APIs and a Cesium-based demo interface with 3D object upload and simple physics. However, much of the code is scaffold or proof-of-concept; no mature robotics middleware (e.g. ROS connectivity, robust sensor models, or industrial-strength physics) is yet present. The existing architecture (folders like `/backend`, `/physics`, `/ai_integration`, `/3d_objects`) suggests plans for modular components, but no strong enforcement of abstraction layers is visible.

Proposed Modular Architecture

To evolve into a full-featured cross-domain platform, we recommend a **layered, modular architecture** with clear abstraction between (a) *data/integration*, (b) *simulation engine*, (c) *robot interface*, and (d) *visualization/UI*, plus an *AI/agent* layer. For example, many digital-twin frameworks use a **“context broker”** at the heart to track real-world state and synchronize the virtual model. In FIWARE’s reference architecture, a context broker manages digital-twin entities (robots, sensors, environment) and converts between IoT protocols and the platform’s data model. We suggest adopting a similar pattern: a central message/contexts hub (could be based on ROS 2 DDS, MQTT broker, or NGSI/REST) that stores device states, telemetry, and configuration. On top of this layer, **simulation services** (physics engine, sensor models) generate synthetic data, while **hardware interfaces** (ROS bridges, firmware connectors) feed data into the hub. This separation ensures that the same state representation can drive both the simulation and the live system.

Key abstraction layers should include:

- **Sensor/Actuator Abstraction:** Define unified interfaces for each sensor/actuator type (cameras, LiDARs, IMUs, GPS, controllers, motors). For example, physics engines can emit synthetic LiDAR points or images using a common API. Real robots can plug in via ROS drivers or MQTT topics using the same API contract. This makes it possible to swap real or simulated hardware transparently.

- **Physics Engine Layer:** Integrate a robust 3D physics simulator (e.g. use Gazebo/Ignition, or a high-performance engine like Bullet/PhysX). The engine should handle rigid-body kinematics/dynamics, collisions, material properties, and sensor physics. As one reviewer notes, Gazebo (Ignition) is an open-source robotics simulator that "simulates robot kinematics, dynamics, sensors, and interactions with conveyors, fixtures and AGVs in 3D". We can leverage Gazebo or CoppeliaSim for robot-cell and vehicle simulations. Alternatively, Unity/Unreal (via their physics subsystems) can be used for high-fidelity scenarios, though these engines are more heavyweight and non-web-native.

- **Hardware/Robot Layer:** Build adapters (e.g. ROS/ROS2 nodes, MQTT clients, OPC-UA modules) to connect physical robots and IoT devices. These convert native protocols into the platform's context format. For example, ROS1/ROS2 topics can be bridged to the central broker (tools like ROSBridge or rclpy can publish JSON over WebSockets or REST). MQTT is a lightweight IoT transport widely used for digital twins, it "provides a centralized communication channel for digital twin applications enabling real-time monitoring with scalable and reliable data". The platform should support at least ROS messages and MQTT, with optional TCP/UDP or CAN bus links for special hardware.

- **Visualization/UI Layer:** Provide a modular UI layer for 3D visualization, dashboards, and control panels. This could be a web-based front end (e.g. CesiumJS or WebGL) and/or native VR/AR apps. The front end would subscribe to the context broker to render robots, environments, and telemetry in real time. CesiumJS is well-suited for geospatial/global scenarios: it is an open-source JS library for high-performance 3D globes and maps, enabling interactive visualization of dynamic geospatial data. However, for indoor or detailed industrial scenes, alternatives like **Unity** or **Unreal Engine** offer far richer graphics and physics. Unity and Unreal can import CAD/BIM data directly and simulate complex environments; for example, an industrial digital-twin built on Unreal enabled city-scale civil models to run at interactive frame rates. We propose a hybrid approach: **keep CesiumJS for geo/mapping tasks** (e.g. farm fields, city models) and provide plugins or connectors to Unity/Unreal for close-up simulations requiring photorealism and advanced physics (e.g. warehouse interiors, vehicle simulators). WebGL frameworks (e.g. Three.js or Babylon.js) and emerging WebGPU technology can complement by offering browser-based 3D without requiring full game engines.

Advanced Simulation and Domain Support

The platform must serve **diverse domains** (agriculture, autonomous vehicles, industrial robots, environmental monitoring). This means providing example modules or plugins for each:

- **Agriculture Robots:** Support field robot scenarios with GPS mapping and variable terrain. For example, a multi-agent farm simulator (Agri-RO5) used Unity3D + ROS to coordinate multiple agri-robots for tasks like plowing and harvesting. The platform can include realistic crop models, multi-robot coordination, and weather effects.

- **Autonomous Vehicles:** Include road network maps (via Cesium/OSM), vehicle dynamics models, traffic simulation, and sensor suites (cameras, radar, LiDAR). Leverage open datasets (OpenStreetMap, lane network) and physics libraries (e.g. Bullet or car dynamics plugins).

- **Industrial Robotics:** Provide modular cell simulators (robot arms with conveyors and shelves). Gazebo/Webots are ideal here: they can simulate multiple robotic arms with cameras and force sensors. Engineers should be able to design a workcell (e.g. via URDF/SDF) and test collision, cycle-time, and PLC logic in simulation before deployment.

- **Environmental & Infrastructure Monitoring:** Model sensor networks (drones, satellites, IoT sensors) over terrain and city models. Digital twins of smart cities often integrate GIS data and simulation of environmental factors. Cesium's "3D Tiles" streaming format can handle large city/building models efficiently. The platform could also simulate water/gas/traffic flow via physics-based models and allow IoT data ingestion.

Cross-domain extensibility is achieved by a plugin architecture: third parties could develop **"scenario plugins"** (e.g. an orchard with robot tree pruners, or a distribution center with AMRs). Each plugin would register its models and parameters (using a common schema) and the platform would handle rendering and simulation.

Visualization & UI Evaluation

CesiumJS – Already in use – is excellent for global geospatial visualization. It "powers the spatial side of digital twins by managing location, terrain, imagery and 3D content as one synchronized whole". Cesium automatically handles map projections, precision, and level-of-detail, which avoids scale/alignment errors. Its tile-streaming (3D Tiles) supports huge models (entire cities or farmland) with minimal load times. For broad environmental contexts (e.g. agriculture fields, city maps, topography) Cesium should be retained as the base engine.

Unity/Unreal – These game engines offer **photorealistic rendering and advanced physics**. For example, Unreal Engine achieves real-time global illumination and handles billions of polygons with instancing and virtual textures. A mobile workstation can run interactive city-scale models at 30+ FPS by using LOD and streaming (as shown in a Philadelphia I-95 city model demo). Unreal also natively imports BIM/CAD (Revit, IFC) so engineering models need no conversion. Critically, Unreal can simulate beyond visuals: its physics, lighting, and logic systems allow scenario testing (evacuation, traffic, construction phasing) that shifts a twin from "impressive to indispensable". We suggest adding Unity/Unreal support as **alternative front-ends**: developers could export the Cesium-managed geodata into these engines for high-fidelity sessions, or run these engines standalone for specific tasks. The choice can be modular: the core platform provides data feeds (maps, telemetry), and the user can plug in the engine of choice for rendering.

WebGL/Three.js/WebGPU – For browser-based simplicity, lightweight simulations can use Three.js or WebGPU (the emerging Web graphics API). These require more custom coding for GIS or physics, but are fully in-browser. In the long term, WebGPU can harness GPU acceleration for large-scale sim without a native client. A hybrid approach (Cesium+Three.js overlays, or Cesium for geodata + WebGL for robots) can be useful.

Multi-Robot & Agent Modeling

The platform should natively support **multi-agent simulation**. For instance, an agricultural scenario might involve many robots coordinating tasks; the Agri-RO5 study showed a ROS+Unity+SPADE agent framework that smoothly coordinates a robot fleet in simulation. Each robot can be modeled as an autonomous agent (with its own AI/logic), interacting via the platform's message bus. Similarly, warehouse or drone swarms should be supported. Pre-built algorithms (e.g. task-allocation, path-planning) can be offered as services. We recommend exposing a plugin interface for *behavior modules*, so researchers can test new RL/ML strategies in the simulated environment.

User Classes and Experience

Different user groups have varying needs:

- **Roboticists/Engineers:** Require full control of sensors and code. Provide a **programmatic API** (Python/ROS nodes, scripts) and detailed telemetry (e.g. `/api/telemetry`). Advanced users should be able to write custom control code or physics modules.

- **Researchers/Students:** Need reproducibility and flexibility. Offer high-level scripting (Jupyter notebooks, or visual scenario editors) and a library of examples. Provide sandboxed environments to test algorithms safely. A modular UI where they can drag-and-drop robots, sensors, and write logic (e.g. Blockly or node-RED style interface)

would help lower the entry barrier.

- **Non-Technical Users:** Want simple, intuitive interactions (e.g. teachers or planners). Offer **preset scenarios and graphical dashboards**. For example, a drag-and-drop interface could let a user place a farm robot on a field map and press “Run”. Provide visual controls (sliders for speed, checkboxes for behaviors) instead of code. Consider natural-language queries (leveraging LLMs) or question-answering on simulation results. As an illustration, the humanoid Pepper robot below would be used in social/educational settings; the platform should simulate Pepper’s behaviors with a few clicks so users can focus on application rather than coding.

To accommodate all users, the design should be **modular**: core simulation modules underlie a spectrum of UI layers and APIs. A “core engine” can run headless with command-line or API control for experts, while a web GUI provides wizards for others. Feature flags or user modes (basic vs advanced) can hide complexity.

Folder & Module Structure

Our team and community has proposed reorganizing into a clear **monorepo** or multi-repo scheme:

- `core/`: core services (context broker, physics server, data storage)
- `sim/`: simulation scenarios, physics engine integration, sensor models
- `robo_iface/`: connectors for ROS/MQTT/OPC-UA, robot drivers, protocol adapters
- `ui/`: front-end code (web apps, Cesium layers, dashboards)
- `ai/`: AI algorithms (motion planning, learning agents) and integration with LLMs or ML libraries
- `plugins/`: example plugins for domains (e.g. `plugins/agri`, `plugins/warehouse`) containing scenario definitions and assets
- `docs/`, `tests/`, `ci/`: documentation, automated tests, and CI/CD pipelines

Each module should have its own API boundary (e.g. via messages or REST endpoints). Shared definitions (e.g. JSON schemas for sensor data) live in a `common/` folder. The repository should enforce clean interfaces so that new plugins or modules can be added without modifying core code.

Plugin and Extension Strategy

The platform must be extensible. We suggest a **plugin architecture** akin to ROS nodes or Unity asset packages. Plugins can be for new robot types, new sensors, new algorithms, or new UI widgets. For example, a sensor plugin could register a driver and data format (e.g. “thermal camera model”), and the core UI would auto-generate visualization options. A physics plugin could add a novel vehicle dynamic model. Use a dynamic loader (e.g. Node.js `require` or Python’s entry points) so third-party modules can be discovered at runtime. Provide a well-documented plugin API and template. An extension marketplace (even if just a GitHub org) can help share community modules.

Deployment (Local, Cloud, Hybrid)

Local Deployment: Good for development and connecting to real robots. The platform should run on desktops or local servers (via Docker containers) so engineers can interface hardware and tune in real time. Local simulators (with GPU acceleration) allow for fast iteration.
Cloud Deployment: Useful for large-scale simulations and collaboration. We envision a cloud version where the backend and heavy simulation run on servers or clusters. Users could log in via web and run experiments on-demand. GPU instances or WebGPU support could accelerate large physics simulations. Containerization (Docker/Kubernetes) will ease deployment. We should also enable **edge/hybrid**: for example, a cloud UI controlling a locally hosted simulation or robot, or syncing sensor data to the cloud. Using MQTT or DDS over internet can bridge local/cloud.

If commercialization is considered, offering a hosted SaaS version could be an option: users subscribe to cloud compute for large-scale sims, while core software remains open-source (Apache-2.0).

APIs and Protocols

The platform should expose rich APIs for integration. The existing REST endpoints (like `/health`, `/api/geo/area`, etc.) are a good start, but more are needed. We recommend:

- **Real-time Data APIs:** WebSocket or MQTT streams for sensor data and robot states (to support live dashboards).
- **Robot Control APIs:** e.g. WebRTC or ROS Bridge for remote robot teleoperation.
- **Standard Data Models:** Adopt or define JSON schemas (or NGSI-LD entities) for robot positions, sensor readings, and object states. Using standards (ROS message types, OPC-UA info models, or NGSI IoT models) will ease interoperability.

- **Plugin APIs:** Method hooks (callbacks) for plugin initialization, command invocation, etc.

- **Scripting/SDK:** Provide an SDK (e.g. Python package) so users can script simulations (set up scenes, run trials, collect results).

For communication, we should support common robotics/IoT protocols:
ROS/ROS2 (via DDS), MQTT, and optionally AMQP or Kafka for telemetry. As one guide notes, MQTT is lightweight and ensures data integrity with QoS features – ideal for sensor telemetry and digital-twin synchronization. ROS 2's DDS offers real-time QoS if needed. For higher-level orchestration, REST/gRPC can manage tasks and queries.

The Motivation, Architecture, and Community Governance of an Open-Source Web-Based Robotics Platform That Was Designed to be Made for the International Engineering Community

The development of robotics and cyber-physical systems has historically been constrained by fragmented toolchains, proprietary software ecosystems, and limited accessibility. *Robotics Platforms* is an open-source, web-based robotics and digital-twin framework founded and led by **Paul George Savluc** to address these structural limitations. This paper presents the philosophical, technical, and socio-technical motivations behind the platform, examines the rationale for its web-based and open-source architecture, and analyzes how community collaboration functions as a core design principle rather than a secondary feature. The work further situates the platform within a broader shift toward democratized physical intelligence and shared stewardship of robotics infrastructure.

Robotics and digital-twin technologies are increasingly central to modern engineering, autonomous systems, and intelligent infrastructure. Despite rapid advances in hardware and artificial intelligence, development workflows remain fragmented across incompatible tools and closed platforms. These structural inefficiencies slow innovation, limit collaboration, and create high barriers to entry for new researchers, students, and small organizations.

Robotics Platforms was conceived as a response to these systemic challenges. Rather than functioning as a conventional commercial software product with a closed roadmap, it was designed as an open, extensible environment in which robotics, simulation, artificial intelligence, and physical context converge within a shared, inspectable system.

Fragmentation as a Structural Constraint in Robotics Development

A central motivation for the platform is the pervasive fragmentation in contemporary robotics workflows. Current development environments frequently separate mechanical and electronic

design from simulation, simulation from real-world geography, and artificial intelligence from physical constraints. In addition, experimentation is often restricted by high costs and vendor-controlled ecosystems.

As a result, innovation frequently stalls not due to conceptual limitations, but because integration across tools is costly, fragile, or infeasible. Based on extensive experience across embedded systems, artificial intelligence, simulation, and hardware research and development, the project's founder identified fragmentation as a primary bottleneck to progress.

Robotics Platforms was therefore designed as a unifying layer in which hardware, software, physics, geographic context, and AI reasoning coexist within a single coherent framework. This integration enables holistic reasoning about systems rather than isolated optimization of individual components.

Rationale for a Web-Based Architecture

The decision to implement Robotics Platforms as a web-based system reflects a philosophical stance as much as a technical one. Traditional robotics software environments typically assume dedicated workstations, proprietary licenses, local installations, and long-term vendor dependence. These assumptions implicitly restrict participation and slow collaboration.

By contrast, a web-based platform assumes minimal prerequisites: a standards-compliant browser, open protocols, and globally distributed contributors. Operating within the browser enables simulations to be shared as links rather than files, transforms collaboration from asynchronous hand-offs to real-time interaction, and reduces onboarding from weeks to minutes.

This architectural approach aligns robotics development with modern software engineering practices, emphasizing accessibility, rapid iteration, and collaborative workflows over isolated, institution-bound toolchains.

Digital Twins as the Primary Object of Reasoning

Within Robotics Platforms, digital twins are not treated as auxiliary visualization tools or abstract marketing constructs. Instead, they function as core computational entities. Each digital twin represents a living state machine corresponding to a physical system, capturing its configuration, dynamics, and interactions over time.

These twins serve simultaneously as testable hypotheses for system behavior and as bridges between simulation and deployment. Every robot, sensor, vehicle, or infrastructure element possesses an explicit state that can be simulated, modified, replayed, or analyzed. Artificial intelligence components reason about these stateful twins rather than operating solely on abstract textual prompts.

This design enables AI systems to incorporate physical consequences into decision-making processes, supporting intelligence grounded in physics and embodiment rather than language alone.

Open Source as a Foundational Requirement

The open-source nature of Robotics Platforms is not incidental; it is a necessary condition for achieving its objectives. Closed robotics platforms inherently obscure physics assumptions, AI decision logic, and safety constraints while binding users to vendor-controlled lifecycles.

An open-source robotics platform, by contrast, enables inspectable physics models, auditable AI behavior, community-validated safety practices, and long-term survivability independent of any single organization. The guiding principle underlying this choice is straightforward: systems capable of influencing the physical world must be transparent to those who depend on them.

In this context, open source is not primarily about cost reduction. It is about trust, continuity, and the accumulation of shared intelligence across generations of contributors.

Multi-Layered Community Collaboration

Community collaboration within Robotics Platforms extends beyond code contributions and operates across multiple interrelated layers.

Technical Collaboration

The platform’s modular architecture allows contributors to work independently on physics models, AI reasoning pipelines, user interfaces, sensor abstractions, rendering optimizations, or hardware adapters without interfering with one another. This modularity lowers coordination overhead and encourages parallel innovation.

Knowledge Collaboration

The repository functions as a public knowledge commons. Design decisions, assumptions, trade-offs, and failures are documented in code and discussion rather than hidden in proprietary toolchains or internal documentation. This public accumulation of knowledge ensures that learning compounds over time instead of being lost.

Ethical and Safety Collaboration

Open development enables peer review of simulation assumptions and public scrutiny of AI decision pathways. As autonomous systems proliferate and control loops tighten, early detection of unsafe patterns becomes critical. In this model, the community itself acts as a distributed safety mechanism rather than relegating safety to post-deployment audits.

Implications of an Open-Source Web-Based Robotics Platform

The emergence of an open, web-based robotics platform represents a broader shift in how robotics infrastructure is conceived and governed.

Historically, robotics has been expensive, institution-centric, and slow to iterate. Open, browser-based systems reduce scarcity by enabling exploration, modification, and prototyping without prohibitive capital investment. They transform static tools into living systems that evolve through continuous contribution and adaptation.

Governance also shifts from centralized authority to shared stewardship. Decisions about capabilities, constraints, and evolution are made transparently and negotiated collectively, increasing resilience and long-term viability.

Broader Vision and Future Trajectory

Robotics Platforms forms part of a larger effort to democratize physical intelligence. This effort does not seek to simplify robotics by concealing complexity, but rather to expose complexity in a structured, inspectable manner. By making reasoning explicit and collaboration normative, the platform invites a diverse community of engineers, researchers, students, and builders to co-author the future of intelligent physical systems.

In this sense, the project is less concerned with the immediate state of robotics technology and more focused on how humanity designs, governs, and shares the systems that increasingly shape the physical world.

Community, Governance, and Roadmap

As an open-source project, clear governance will attract contributors. We recommend forming a **maintainer team** (perhaps via a GitHub Organization) and documenting roles in a GOVERNANCE.md. Early on, the core authors should retain commit rights, but use pull requests and issue templates to encourage contributions. As the guide suggests, list maintainers in the README/CONTRIBUTORS file and allow others to become maintainers after sustained contributions. Use continuous integration for testing (as already done) and require passing checks for merges.

A modular plugin architecture will let contributors add domain-specific features without touching the core. We should maintain an official extension registry or documentation of approved plugins. Licensing under Apache-2.0 keeps the code open while allowing commercial services. For a potential commercialization path, consider offering consulting, enterprise support, or proprietary modules (e.g. high-fidelity sensor models) under dual-license terms.

In summary, evolving **Robotics-Platforms** into a full simulation and digital twin ecosystem requires building out robust simulation kernels, clear abstraction layers, flexible UI options, and strong community processes. By leveraging existing open-source tools (Gazebo, Cesium, ROS, etc.) and adopting a plugin-based design, the platform can support varied applications – from farm robots to smart factories – while remaining accessible to all user classes.

Security and Sandbox

Running user-provided code (scripts, plugins, AI models) poses risks. We must implement sandboxing and security layers. Suggestions include:

- **Container Isolation:** Run simulations and plugins in containers or microVMs (e.g. Docker, Firecracker) to prevent rogue code from escaping.

- **WebAssembly Sandboxing:** For web-exposed scripting (e.g. browser simulation scripts), use a WASM-based engine (like QuickJS in WASM) to safely run JS code in a constrained environment.

- **Authentication/Authorization:** Secure all APIs with token-based auth. Implement role-based access: e.g. “viewer” vs “operator” vs “admin.” The context broker should enforce access policies on who can read or command each digital twin.

- **Input Validation:** Rigorously validate all incoming data (geo coords, sensor inputs) to avoid injection attacks.

- **Audit and Monitoring:** Log all actions for audit, and provide “kill switch” controls to pause or stop simulations in emergencies.

By design, any plugin or script should run with least privilege. For instance, third-party AI agents should only interact through predefined channels, not execute arbitrary system commands. As the platform may connect to real hardware, it must enforce safe-operating envelopes (e.g. virtual fences, emergency-stop logic)

Robotics Platforms demonstrates how an open-source repository can combine AI, digital twins and immersive 3-D simulation to support hardware engineering and robotics development. The project integrates a FastAPI/Flask backend, a CesiumJS front end, basic physics, and early AI functions to provide a proof-of-concept digital-twin platform. Its modular design, separating services, simulation state, physics modules and AI integration, makes it suitable for incremental contributions.

As digital-twin technology continues to mature, open-source collaborations will play an important role in standardising models, sharing best practices and fostering interoperability. By contributing to projects like Robotics Platforms, developers can help shape the future of

robotics, simulation and hardware design. Whether through improving the AI layer, enhancing physics realism, expanding asset libraries or documenting workflows, the community's participation will make the platform more robust and versatile, ultimately driving innovation in robotics and digital-twin engineering.

Code Appendix:

Digital_Twin.html

```
<!DOCTYPE html>

<html lang="en">

 <head>

   <meta charset="utf-8" />

   <title>OPENQQUANTIFY Digital Twin IDE</title>

   <script
src="https://cesium.com/downloads/cesiumjs/releases/1.121/Build/Cesium/Cesium.j
s"></script>

   <link


href="https://cesium.com/downloads/cesiumjs/releases/1.121/Build/Cesium/Widgets
/widgets.css"

     rel="stylesheet"

   />

   <link

     rel="stylesheet"


href="https://cdn.jsdelivr.net/npm/spectrum-colorpicker2/dist/spectrum.min.css"

   />

   <script
src="https://cdn.jsdelivr.net/npm/spectrum-colorpicker2/dist/spectrum.min.js"><
/script>

   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

   <link

     rel="stylesheet"
```

```
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css
"

   />

   <link

     rel="stylesheet"


href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/codemirror.min.c
ss"

   />

   <link

     rel="stylesheet"


href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/theme/dracula.mi
n.css"

   />

   <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/codemirror.min.js
"></script>

   <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/mode/javascript/j
avascript.min.js"></script>

   <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/addon/edit/matchb
rackets.min.js"></script>

   <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/addon/edit/closeb
rackets.min.js"></script>

   <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/addon/comment/com
ment.min.js"></script>

   <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/addon/hint/javasc
ript-hint.min.js"></script>
```

```
   <script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.2/addon/hint/show-h
int.min.js"></script>

   <style>

     /* Global styles */

     :root {

       --primary: #00a3cc;

       --primary-dark: #0088a8;

       --primary-light: #00b8e6;

       --bg-dark: #121212;

       --bg-darker: #0a0a0a;

       --bg-light: #1e1e1e;

       --bg-lighter: #2a2a2a;

       --text-primary: #ffffff;

       --text-secondary: #b0b0b0;

       --border-dark: #333333;

       --border-light: #444444;

       --success: #2ecc71;

       --warning: #f39c12;

       --danger: #e74c3c;

       --info: #3498db;

     }



     * {

       box-sizing: border-box;

     }
```

```
    html,
    body,
    #cesiumContainer {
      width: 100%;
      height: 100%;
      margin: 0;
      padding: 0;
      overflow: hidden;
      font-family: 'Segoe UI', 'Roboto', 'Oxygen', 'Ubuntu', 'Cantarell',
'Fira Sans',
        'Droid Sans', sans-serif;
      background-color: var(--bg-dark);
      color: var(--text-primary);
    }

    /* Main layout */
    .app-container {
      display: flex;
      height: 100vh;
      background-color: var(--bg-dark);
    }

    /* Sidebar styles */
    .sidebar {
      width: 300px;
      background-color: var(--bg-darker);
```

```
  border-right: 1px solid var(--border-dark);
  display: flex;
  flex-direction: column;
  transition: all 0.3s ease;
  z-index: 100;
}

.sidebar.collapsed {
  width: 50px;
}

.sidebar-header {
  padding: 15px;
  background-color: var(--bg-dark);
  color: var(--text-primary);
  display: flex;
  justify-content: space-between;
  align-items: center;
  border-bottom: 1px solid var(--border-dark);
}

.sidebar-header h2 {
  margin: 0;
  font-size: 18px;
  font-weight: 600;
  color: var(--primary);
```

```
}

.toggle-sidebar {
  background: none;
  border: none;
  color: var(--text-secondary);
  cursor: pointer;
  font-size: 16px;
  transition: all 0.2s ease;
}

.toggle-sidebar:hover {
  color: var(--primary);
  transform: scale(1.1);
}

/* Tab system */
.sidebar-tabs {
  display: flex;
  border-bottom: 1px solid var(--border-dark);
}

.sidebar-tab {
  padding: 12px 15px;
  cursor: pointer;
  background-color: var(--bg-dark);
```

```
  border-right: 1px solid var(--border-dark);
  font-size: 13px;
  color: var(--text-secondary);
  transition: all 0.2s ease;
  display: flex;
  align-items: center;
  gap: 8px;
}

.sidebar-tab:hover {
  background-color: var(--bg-light);
  color: var(--text-primary);
}

.sidebar-tab.active {
  background-color: var(--bg-darker);
  border-bottom: 2px solid var(--primary);
  color: var(--text-primary);
}

.sidebar-tab i {
  font-size: 14px;
}

/* Tab content */
.tab-content {
```

```
    display: none;
    flex: 1;
    overflow-y: auto;
    padding: 10px;
    background-color: var(--bg-darker);
}

.tab-content.active {
    display: block;
}

/* Function documentation styles */
.function-doc {
    background-color: var(--bg-dark);
    border-radius: 6px;
    padding: 12px;
    margin-bottom: 15px;
    border-left: 3px solid var(--primary);
}

.function-name {
    font-weight: bold;
    color: var(--primary);
    margin-bottom: 5px;
}
```

```
.function-description {
  font-size: 13px;
  color: var(--text-secondary);
  margin-bottom: 8px;
}

.function-params {
  font-size: 12px;
  color: var(--text-secondary);
}

.param-name {
  color: var(--info);
}

/* File explorer */
.file-explorer {
  list-style: none;
  padding: 0;
  margin: 0;
}

.file-item {
  padding: 10px 12px;
  cursor: pointer;
  display: flex;
```

```
    align-items: center;
    border-radius: 4px;
    color: var(--text-secondary);
    transition: all 0.2s ease;
    margin-bottom: 2px;
  }

  .file-item:hover {
    background-color: var(--bg-light);
    color: var(--text-primary);
  }

  .file-item i {
    margin-right: 10px;
    font-size: 13px;
    color: var(--text-secondary);
    transition: all 0.2s ease;
  }

  .file-item.active {
    background-color: var(--bg-light);
    color: var(--primary);
  }

  .file-item.active i {
    color: var(--primary);
```

```
}

/* Main content area */
.main-content {
  flex: 1;
  display: flex;
  flex-direction: column;
  position: relative;
  background-color: var(--bg-dark);
}

/* Editor layout */
.editor-container {
  display: flex;
  flex: 1;
  overflow: hidden;
}

.code-editor {
  flex: 1;
  display: flex;
  flex-direction: column;
  border-right: 1px solid var(--border-dark);
}

.editor-toolbar {
```

```
  padding: 10px 15px;
  background-color: var(--bg-dark);
  border-bottom: 1px solid var(--border-dark);
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.editor-title {
  font-size: 14px;
  margin: 0;
  color: var(--text-secondary);
  font-weight: 500;
}

.editor-actions {
  display: flex;
  gap: 8px;
}

.CodeMirror {
  height: 100% !important;
  font-size: 14px;
  background-color: var(--bg-darker) !important;
}
```

```
.CodeMirror-gutters {
  background-color: var(--bg-darker) !important;
  border-right: 1px solid var(--border-dark) !important;
}

/* Preview pane */
.preview-pane {
  flex: 1;
  display: flex;
  flex-direction: column;
  position: relative;
}

/* Top toolbar */
.top-toolbar {
  padding: 12px 15px;
  background-color: var(--bg-dark);
  border-bottom: 1px solid var(--border-dark);
  display: flex;
  justify-content: space-between;
  align-items: center;
  z-index: 10;
}

.app-title {
  font-size: 20px;
```

```
  font-weight: 700;
  color: var(--primary);
  margin: 0;
  letter-spacing: 0.5px;
}

.toolbar-controls {
  display: flex;
  gap: 10px;
}

/* Buttons */
.btn {
  padding: 8px 14px;
  border-radius: 6px;
  border: none;
  background-color: var(--primary);
  color: white;
  cursor: pointer;
  font-size: 13px;
  font-weight: 500;
  transition: all 0.2s ease;
  display: flex;
  align-items: center;
  gap: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
```

```
}

.btn:hover {
  background-color: var(--primary-dark);
  transform: translateY(-1px);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
}

.btn:active {
  transform: translateY(0);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
}

.btn-outline {
  background-color: transparent;
  border: 1px solid var(--primary);
  color: var(--primary);
}

.btn-outline:hover {
  background-color: rgba(0, 163, 204, 0.1);
}

.btn-sm {
  padding: 6px 10px;
  font-size: 12px;
```

```
}

/* Console */
.console-container {
  height: 200px;
  background-color: var(--bg-darker);
  border-top: 1px solid var(--border-dark);
  display: flex;
  flex-direction: column;
}

.console-header {
  padding: 8px 12px;
  background-color: var(--bg-dark);
  border-bottom: 1px solid var(--border-dark);
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.console-title {
  font-size: 13px;
  margin: 0;
  color: var(--text-secondary);
  font-weight: 500;
}
```

```
.console-clear {
  background: none;
  border: none;
  color: var(--text-secondary);
  cursor: pointer;
  font-size: 12px;
  transition: all 0.2s ease;
}

.console-clear:hover {
  color: var(--primary);
}

.console-content {
  flex: 1;
  overflow-y: auto;
  padding: 10px;
  font-family: 'Fira Code', 'Consolas', 'Monaco', monospace;
  font-size: 13px;
  white-space: pre-wrap;
  color: var(--text-secondary);
  line-height: 1.5;
}

/* Sensor categories */
```

```
.sensor-category {
  margin-bottom: 15px;
  background-color: var(--bg-dark);
  border-radius: 6px;
  overflow: hidden;
  border: 1px solid var(--border-dark);
}

.category-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 12px;
  background-color: var(--bg-dark);
  color: var(--text-primary);
  cursor: pointer;
  font-weight: 500;
  transition: all 0.2s ease;
}

.category-header:hover {
  background-color: var(--bg-light);
}

.category-header i {
  transition: transform 0.2s ease;
```

```
}

.sensor-items {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  gap: 10px;
  padding: 10px;
  background-color: var(--bg-darker);
}

.sensor-item {
  display: flex;
  flex-direction: column;
  align-items: center;
  padding: 12px;
  background-color: var(--bg-dark);
  border: 1px solid var(--border-dark);
  border-radius: 6px;
  cursor: grab;
  transition: all 0.2s ease;
}

.sensor-item:hover {
  border-color: var(--primary);
  box-shadow: 0 4px 12px rgba(0, 163, 204, 0.3);
  transform: translateY(-2px);
```

```
}

.sensor-item i {
  font-size: 20px;
  margin-bottom: 8px;
  color: var(--primary);
}

.sensor-item span {
  font-size: 12px;
  text-align: center;
  color: var(--text-secondary);
}

/* Properties panel */
.properties-panel {
  position: absolute;
  right: 20px;
  top: 60px;
  width: 320px;
  background-color: var(--bg-darker);
  border-radius: 8px;
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.3);
  border: 1px solid var(--border-dark);
  z-index: 100;
  display: none;
```

```
  overflow: hidden;
}

.properties-header {
  padding: 12px;
  background-color: var(--bg-dark);
  color: white;
  display: flex;
  justify-content: space-between;
  align-items: center;
  border-bottom: 1px solid var(--border-dark);
}

.properties-title {
  margin: 0;
  font-size: 14px;
  font-weight: 500;
}

.properties-close {
  background: none;
  border: none;
  color: var(--text-secondary);
  cursor: pointer;
  transition: all 0.2s ease;
}
```

```
.properties-close:hover {
  color: var(--primary);
  transform: scale(1.1);
}

.properties-content {
  padding: 15px;
}

.property-row {
  margin-bottom: 15px;
}

.property-label {
  display: block;
  margin-bottom: 6px;
  font-size: 13px;
  color: var(--text-secondary);
  font-weight: 500;
}

.property-input {
  width: 100%;
  padding: 8px 12px;
  background-color: var(--bg-dark);
```

```
  border: 1px solid var(--border-dark);
  border-radius: 6px;
  color: var(--text-primary);
  transition: all 0.2s ease;
}

.property-input:focus {
  outline: none;
  border-color: var(--primary);
  box-shadow: 0 0 0 2px rgba(0, 163, 204, 0.2);
}

/* Upload modal */
.upload-modal {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.8);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1000;
  display: none;
}
```

```
.upload-content {
  background-color: var(--bg-darker);
  border-radius: 8px;
  padding: 20px;
  width: 500px;
  max-width: 90%;
  border: 1px solid var(--primary);
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.4);
}

.upload-header {
  margin-bottom: 20px;
  color: var(--text-primary);
  font-size: 18px;
  font-weight: 500;
}

.upload-form {
  display: flex;
  flex-direction: column;
  gap: 15px;
}

.form-group {
  display: flex;
```

```
    flex-direction: column;
  }

  .form-label {
    margin-bottom: 6px;
    color: var(--text-secondary);
    font-size: 13px;
    font-weight: 500;
  }

  .form-input {
    padding: 10px 12px;
    background-color: var(--bg-dark);
    border: 1px solid var(--border-dark);
    border-radius: 6px;
    color: var(--text-primary);
    transition: all 0.2s ease;
  }

  .form-input:focus {
    outline: none;
    border-color: var(--primary);
    box-shadow: 0 0 0 2px rgba(0, 163, 204, 0.2);
  }

  .upload-actions {
```

```
  display: flex;
  justify-content: flex-end;
  gap: 10px;
  margin-top: 20px;
}

/* AI Assistant Panel */
.ai-assistant-panel {
  position: absolute;
  right: 20px;
  bottom: 20px;
  width: 320px;
  background-color: var(--bg-darker);
  border-radius: 8px;
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.3);
  border: 1px solid var(--primary);
  z-index: 100;
  display: none;
  overflow: hidden;
}

.ai-assistant-header {
  padding: 12px;
  background-color: var(--bg-dark);
  color: white;
  display: flex;
```

```
  justify-content: space-between;
  align-items: center;
  border-bottom: 1px solid var(--border-dark);
}

.ai-assistant-title {
  margin: 0;
  font-size: 14px;
  font-weight: 500;
}

.ai-assistant-close {
  background: none;
  border: none;
  color: var(--text-secondary);
  cursor: pointer;
  transition: all 0.2s ease;
}

.ai-assistant-close:hover {
  color: var(--primary);
  transform: scale(1.1);
}

.ai-assistant-content {
  padding: 15px;
```

```
  max-height: 300px;
  overflow-y: auto;
}

.ai-message {
  margin-bottom: 10px;
  padding: 8px 12px;
  border-radius: 6px;
  font-size: 13px;
  line-height: 1.4;
}

.ai-user {
  background-color: var(--bg-light);
  color: var(--text-primary);
  margin-left: 20px;
  border-left: 2px solid var(--primary);
}

.ai-assistant {
  background-color: var(--bg-dark);
  color: var(--text-primary);
  margin-right: 20px;
  border-right: 2px solid var(--info);
}
```

```
.ai-input-container {
  display: flex;
  padding: 10px;
  border-top: 1px solid var(--border-dark);
  background-color: var(--bg-dark);
}

.ai-input {
  flex: 1;
  padding: 8px 12px;
  background-color: var(--bg-darker);
  border: 1px solid var(--border-dark);
  border-radius: 6px;
  color: var(--text-primary);
  margin-right: 8px;
}

.ai-send {
  padding: 8px 12px;
  background-color: var(--primary);
  border: none;
  border-radius: 6px;
  color: white;
  cursor: pointer;
}
```

```
/* Collapsed sidebar styles */
.sidebar.collapsed .sidebar-header h2,
.sidebar.collapsed .sidebar-tabs,
.sidebar.collapsed .tab-content {
  display: none;
}

.sidebar.collapsed .sidebar-header {
  justify-content: center;
  padding: 15px 0;
}

.sidebar.collapsed {
  width: 50px;
  overflow: hidden;
}

/* Tooltip */
.tooltip {
  position: absolute;
  background-color: rgba(0, 0, 0, 0.9);
  color: white;
  padding: 8px 12px;
  border-radius: 6px;
  font-size: 12px;
  pointer-events: none;
```

```
    z-index: 1000;
    display: none;
    max-width: 200px;
    backdrop-filter: blur(2px);
  }

  /* Scrollbar styling */
  ::-webkit-scrollbar {
    width: 8px;
    height: 8px;
  }

  ::-webkit-scrollbar-track {
    background: var(--bg-dark);
  }

  ::-webkit-scrollbar-thumb {
    background: var(--border-dark);
    border-radius: 4px;
  }

  ::-webkit-scrollbar-thumb:hover {
    background: var(--border-light);
  }

  /* Responsive adjustments */
```

```
@media (max-width: 1200px) {
  .sidebar {
    width: 250px;
  }
}

@media (max-width: 992px) {
  .editor-container {
    flex-direction: column;
  }

  .code-editor,
  .preview-pane {
    height: 50%;
  }
}

/* Animation */
@keyframes fadeIn {
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}
```

```
    .fade-in {
      animation: fadeIn 0.3s ease-out;
    }

    /* Status colors for console */
    .log-info {
      color: var(--info);
    }

    .log-success {
      color: var(--success);
    }

    .log-warning {
      color: var(--warning);
    }

    .log-error {
      color: var(--danger);
    }
  </style>
</head>
<body>
  <div class="app-container">
    <!-- Sidebar -->
```

```
    <div class="sidebar" id="sidebar">

      <div class="sidebar-header">

        <h2>Digital Twin IDE</h2>

        <button class="toggle-sidebar" id="toggleSidebar">

          <i class="fas fa-chevron-left"></i>

        </button>

      </div>



      <div class="sidebar-tabs">

        <div class="sidebar-tab active" data-tab="explorer">

          <i class="fas fa-folder-open"></i> Explorer

        </div>

        <div class="sidebar-tab" data-tab="entities"><i class="fas
fa-cubes"></i> Entities</div>

        <div class="sidebar-tab" data-tab="sensors">

          <i class="fas fa-satellite-dish"></i> Sensors

        </div>

        <div class="sidebar-tab" data-tab="functions"><i class="fas
fa-book"></i> Functions</div>

      </div>



      <!-- Explorer Tab -->

      <div class="tab-content active" id="explorer-tab">

        <ul class="file-explorer">

          <li class="file-item active" data-file="main.js">

            <i class="fas fa-file-code"></i> main.js

          </li>
```

```
          <li class="file-item" data-file="scene.js">
            <i class="fas fa-file-code"></i> scene.js
          </li>
          <li class="file-item" data-file="entities.js">
            <i class="fas fa-file-code"></i> entities.js
          </li>
          <li class="file-item" data-file="sensors.js">
            <i class="fas fa-file-code"></i> sensors.js
          </li>
        </ul>

        <div style="padding: 10px; border-top: 1px solid #e1e5eb">
          <button class="btn btn-sm" id="uploadModelBtn" style="width: 100%">
            <i class="fas fa-upload"></i> Upload 3D Model
          </button>
        </div>
      </div>

      <!-- Entities Tab -->
      <div class="tab-content" id="entities-tab">
        <div style="padding: 10px">
          <input
            type="text"
            class="form-input"
            placeholder="Search entities..."
            style="margin-bottom: 10px"
```

```
          />
          <div id="entities-list">
            <!-- Entities will be populated here -->
          </div>
        </div>
      </div>

      <!-- Sensors Tab -->
      <div class="tab-content" id="sensors-tab">
        <div style="padding: 10px">
          <div class="sensor-category">
            <div class="category-header">
              <span><i class="fas fa-ruler-combined"></i> Distance
Sensors</span>
              <i class="fas fa-chevron-down"></i>
            </div>
            <div class="sensor-items">
              <div
                class="sensor-item"
                draggable="true"
                data-sensor-type="Ultrasonic"
                data-sensor-shape="cone"
              >
                <i class="fas fa-broadcast-tower"></i>
                <span>Ultrasonic</span>
              </div>
```

```
            <div
              class="sensor-item"
              draggable="true"
              data-sensor-type="Radar"
              data-sensor-shape="cone"
            >
              <i class="fas fa-satellite"></i>
              <span>Radar</span>
            </div>
            <div
              class="sensor-item"
              draggable="true"
              data-sensor-type="LiDAR"
              data-sensor-shape="cone"
            >
              <i class="fas fa-laser"></i>
              <span>LiDAR</span>
            </div>
            <div
              class="sensor-item"
              draggable="true"
              data-sensor-type="Proximity"
              data-sensor-shape="sphere"
            >
              <i class="fas fa-expand"></i>
              <span>Proximity</span>
```

```
      </div>
    </div>
  </div>

  <div class="sensor-category">
    <div class="category-header">
      <span><i class="fas fa-leaf"></i> Environmental</span>
      <i class="fas fa-chevron-down"></i>
    </div>
    <div class="sensor-items">
      <div
        class="sensor-item"
        draggable="true"
        data-sensor-type="Temperature"
        data-sensor-shape="point"
      >
        <i class="fas fa-temperature-high"></i>
        <span>Temperature</span>
      </div>
      <div
        class="sensor-item"
        draggable="true"
        data-sensor-type="Humidity"
        data-sensor-shape="point"
      >
        <i class="fas fa-tint"></i>
```

```
            <span>Humidity</span>
          </div>
          <div
            class="sensor-item"
            draggable="true"
            data-sensor-type="Pressure"
            data-sensor-shape="point"
          >
            <i class="fas fa-tachometer-alt"></i>
            <span>Pressure</span>
          </div>
        </div>
      </div>

      <div class="sensor-category">
        <div class="category-header">
          <span><i class="fas fa-running"></i> Motion Sensors</span>
          <i class="fas fa-chevron-down"></i>
        </div>
        <div class="sensor-items">
          <div
            class="sensor-item"
            draggable="true"
            data-sensor-type="Accelerometer"
            data-sensor-shape="point"
          >
```

```
            <i class="fas fa-arrows-alt"></i>
            <span>Accelerometer</span>
          </div>
          <div
            class="sensor-item"
            draggable="true"
            data-sensor-type="Gyroscope"
            data-sensor-shape="point"
          >
            <i class="fas fa-compass"></i>
            <span>Gyroscope</span>
          </div>
        </div>
      </div>
    </div>
  </div>

  <!-- Functions Documentation Tab -->
  <div class="tab-content" id="functions-tab">
    <div style="padding: 10px">
      <div class="function-doc">
        <div class="function-name">addPoint(lon, lat, height)</div>
        <div class="function-description">
          Adds a point to the scene at the specified coordinates.
        </div>
        <div class="function-params">
```

```
                <div><span class="param-name">lon:</span> Longitude in
degrees</div>

                <div><span class="param-name">lat:</span> Latitude in
degrees</div>

                <div><span class="param-name">height:</span> Height in
meters</div>

              </div>

            </div>



            <div class="function-doc">

              <div class="function-name">addModel(lon, lat, height,
modelUrl)</div>

              <div class="function-description">

                Adds a 3D model to the scene at the specified coordinates.

              </div>

              <div class="function-params">

                <div><span class="param-name">lon:</span> Longitude in
degrees</div>

                <div><span class="param-name">lat:</span> Latitude in
degrees</div>

                <div><span class="param-name">height:</span> Height in
meters</div>

                <div>

                  <span class="param-name">modelUrl:</span> URL or path to the
3D model file

                </div>

              </div>

            </div>



            <div class="function-doc">
```

```
            <div class="function-name">flyTo(lon, lat, height = 1000)</div>

            <div class="function-description">

              Animates the camera to fly to the specified location.

            </div>

            <div class="function-params">

              <div><span class="param-name">lon:</span> Longitude in
degrees</div>

              <div><span class="param-name">lat:</span> Latitude in
degrees</div>

              <div>

                <span class="param-name">height:</span> (Optional) Camera
height in meters

                (default: 1000)

              </div>

            </div>

          </div>



          <div class="function-doc">

            <div class="function-name">initScene()</div>

            <div class="function-description">

              Initializes the scene with default settings and sample entities.

            </div>

          </div>



          <div class="function-doc">

            <div class="function-name">createSensorEntity(sensorParam,
position)</div>
```

```
            <div class="function-description">Creates a sensor entity in the
scene.</div>

            <div class="function-params">

              <div><span class="param-name">sensorParam:</span> Sensor
parameters object</div>

              <div><span class="param-name">position:</span> Cesium Cartesian3
position</div>

            </div>

          </div>



          <div class="function-doc">

            <div class="function-name">makeEntityDraggable(entity)</div>

            <div class="function-description">Makes an entity draggable on the
scene.</div>

            <div class="function-params">

              <div>

                <span class="param-name">entity:</span> The Cesium entity to
make draggable

              </div>

            </div>

          </div>

        </div>

      </div>



      <!-- Main content area -->

      <div class="main-content">

        <!-- Top toolbar -->
```

```
      <div class="top-toolbar">

        <h1 class="app-title">OPENQQUANTIFY Digital Twin IDE</h1>

        <div class="toolbar-controls">

          <button class="btn btn-sm" id="runButton"><i class="fas
fa-play"></i> Run</button>

          <button class="btn btn-sm" id="stopButton"><i class="fas
fa-stop"></i> Stop</button>

          <button class="btn btn-sm" id="aiAssistantButton">

            <i class="fas fa-robot"></i> AI Assistant

          </button>

        </div>

      </div>



      <!-- Editor and Preview -->

      <div class="editor-container">

        <div class="code-editor">

          <div class="editor-toolbar">

            <h3 class="editor-title" id="editorTitle">main.js</h3>

            <div class="editor-actions">

              <button class="btn btn-sm" id="saveButton"><i class="fas
fa-save"></i> Save</button>

            </div>

          </div>

          <textarea id="codeEditor">

// Digital Twin Scripting Environment

// Access the Cesium viewer with the 'viewer' variable
```

```
// Example: Add a simple point to the scene
function addPoint(lon, lat, height) {
 return viewer.entities.add({
   position: Cesium.Cartesian3.fromDegrees(lon, lat, height),
   point: {
     pixelSize: 10,
     color: Cesium.Color.RED,
     outlineColor: Cesium.Color.WHITE,
     outlineWidth: 2
   }
 });
}

// Example: Add a 3D model
function addModel(lon, lat, height, modelUrl) {
 return viewer.entities.add({
   position: Cesium.Cartesian3.fromDegrees(lon, lat, height),
   model: {
     uri: modelUrl,
     minimumPixelSize: 128,
     maximumScale: 20000
   }
 });
}

// Example: Fly to a location
```

```
function flyTo(lon, lat, height = 1000) {
 viewer.camera.flyTo({
   destination: Cesium.Cartesian3.fromDegrees(lon, lat, height),
   orientation: {
     heading: Cesium.Math.toRadians(0.0),
     pitch: Cesium.Math.toRadians(-45.0),
     roll: 0.0
   }
 });
}

// Initialize the scene
function initScene() {
 // Set initial camera position
 flyTo(-74.0060, 40.7128, 5000);

 // Add a sample point
 addPoint(-74.0060, 40.7128, 100);

 console.log("Digital Twin environment initialized");
}

// Run the initialization when the script loads
initScene();</textarea
          >
        </div>
```

```
      <div class="preview-pane">
        <!-- Cesium Container -->
        <div id="cesiumContainer"></div>

        <!-- Console -->
        <div class="console-container">
          <div class="console-header">
            <h3 class="console-title">Console</h3>
            <button class="console-clear" id="clearConsole">
              <i class="fas fa-trash-alt"></i> Clear
            </button>
          </div>
          <div class="console-content" id="consoleOutput"></div>
        </div>
      </div>
    </div>
  </div>

  <!-- Properties Panel -->
  <div class="properties-panel" id="propertiesPanel">
    <div class="properties-header">
      <h3 class="properties-title">Entity Properties</h3>
      <button class="properties-close" id="closePropertiesPanel">
        <i class="fas fa-times"></i>
      </button>
```

```
      </div>

      <div class="properties-content" id="propertiesContent">

        <div class="property-row">

          <label class="property-label">Name</label>

          <input type="text" class="property-input" id="entityName" />

        </div>

        <div class="property-row">

          <label class="property-label">Position (Longitude)</label>

          <input type="number" class="property-input" id="entityLon" />

        </div>

        <div class="property-row">

          <label class="property-label">Position (Latitude)</label>

          <input type="number" class="property-input" id="entityLat" />

        </div>

        <div class="property-row">

          <label class="property-label">Height (meters)</label>

          <input type="number" class="property-input" id="entityHeight" />

        </div>

        <button class="btn btn-sm" id="updateEntity" style="margin-top: 10px">

          <i class="fas fa-save"></i> Update

        </button>

        <button class="btn btn-sm btn-outline" id="deleteEntity"
style="margin-top: 5px">

          <i class="fas fa-trash-alt"></i> Delete

        </button>

      </div>
```

```
    </div>


    <!-- AI Assistant Panel -->
    <div class="ai-assistant-panel" id="aiAssistantPanel">
      <div class="ai-assistant-header">
        <h3 class="ai-assistant-title">AI Assistant</h3>
        <button class="ai-assistant-close" id="closeAIAssistantPanel">
          <i class="fas fa-times"></i>
        </button>
      </div>
      <div class="ai-assistant-content" id="aiAssistantContent">
        <div class="ai-message ai-assistant">
          Hello! I'm your Cesium AI assistant. How can I help you with your
digital twin today?
          You can ask me to:
          <ul>
            <li>Show specific locations (e.g., "Show me the Eiffel
Tower")</li>
            <li>Add entities to the scene</li>
            <li>Explain Cesium functions</li>
            <li>Help with your code</li>
          </ul>
        </div>
      </div>
      <div class="ai-input-container">
        <input
          type="text"
```

```
          class="ai-input"

          id="aiInput"

          placeholder="Ask me anything about Cesium..."

        />

        <button class="ai-send" id="aiSend"><i class="fas
fa-paper-plane"></i></button>

      </div>

    </div>



    <!-- Upload Modal -->

  <div class="upload-modal" id="uploadModal">

    <div class="upload-content">

      <h3 class="upload-header">Upload 3D Model</h3>

      <form class="upload-form" id="uploadForm">

        <div class="form-group">

          <label class="form-label">Model Name</label>

          <input type="text" class="form-input" id="modelName" required>

        </div>

        <div class="form-group">

          <label class="form-label">Model File (GLTF/GLB)</label>

          <input type="file" class="form-input" id="modelFile"
accept=".gltf,.glb" style="display: none;" required>

          <button type="button" class="btn btn-sm" id="selectModelFile"
style="margin-top: 5px;">

            <i class="fas fa-folder-open"></i> Select Model File

          </button>

          <span id="selectedFileName" style="margin-left: 10px; font-size:
12px;">No file selected</span>
```

```
        </div>



        <!-- NEW: persist on server checkbox -->

        <div class="form-group">

          <label class="form-label" style="display:flex; align-items:center;
gap:6px;">

            <input type="checkbox" id="persistOnServer" checked>

            Also upload and persist on server

          </label>

          <span style="font-size: 11px; color: var(--text-secondary);">

            The model will still load locally for this browser session. If
checked, a copy is also stored on the backend under <code>/models</code>.

          </span>

        </div>



        <div class="form-group">

          <label class="form-label">Initial Position (Longitude)</label>

          <input type="number" class="form-input" id="modelLon"
value="-74.0060" step="0.0001">

        </div>

        <div class="form-group">

          <label class="form-label">Initial Position (Latitude)</label>

          <input type="number" class="form-input" id="modelLat"
value="40.7128" step="0.0001">

        </div>

        <div class="form-group">

          <label class="form-label">Initial Height (meters)</label>
```

```
          <input type="number" class="form-input" id="modelHeight"
value="100">
        </div>
        <div class="upload-actions">
          <button type="button" class="btn btn-outline" id="cancelUpload">
            Cancel
          </button>
          <button type="submit" class="btn" id="confirmUpload">
            Upload
          </button>
        </div>
      </form>
    </div>
  </div>

    <!-- Tooltip -->
    <div id="tooltip" class="tooltip"></div>
  </div>
  <script>
    // Initialize Cesium with proper configuration
    // Cesium token is injected from server-side environment when available
    Cesium.Ion.defaultAccessToken = '{{ cesium_ion_token | default("") }}';

    // Global variables
    const viewer = new Cesium.Viewer('cesiumContainer', {
      terrain: Cesium.Terrain.fromWorldTerrain(),
```

```
    animation: false,
    timeline: false,
    shouldAnimate: false,
    infoBox: false,
    selectionIndicator: false,
    shadows: true,
    baseLayerPicker: false,
    navigationHelpButton: false,
    homeButton: false,
    sceneModePicker: false,
    geocoder: false,
    skyAtmosphere: false,
    skyBox: false,
});


// State management
    const state = {
      entities: [],
      selectedEntity: null,
      uploadedModels: {},
      activeHandlers: [],
      isAICodeExecutionEnabled: true,
      currentFile: 'main.js',
      files: {} // REQUIRED
    };
```

```
    // Enhanced scene setup
    function initializeScene() {
      viewer.scene.globe.enableLighting = true;
      viewer.scene.globe.depthTestAgainstTerrain = true;
      viewer.scene.fog.enabled = true;
      viewer.scene.highDynamicRange = true;

      // Load OSM buildings
      Cesium.createOsmBuildingsAsync()
        .then((buildingTileset) =>
viewer.scene.primitives.add(buildingTileset))
        .catch((error) => console.error('Error loading buildings:', error));

      // Set initial camera position
      viewer.camera.setView({
        destination: Cesium.Cartesian3.fromDegrees(-74.006, 40.7128, 2000),
        orientation: {
          heading: Cesium.Math.toRadians(0.0),
          pitch: Cesium.Math.toRadians(-45.0),
          roll: 0.0,
        },
      });
    }

    // Initialize CodeMirror editor
    let codeEditor;
```

```
    (function initInlineEditor() {

      const textarea = document.getElementById('codeEditor');

      if (typeof CodeMirror === 'undefined' || !textarea) {

        console.warn('CodeMirror or #codeEditor not available - using fallback
editor stub');

        codeEditor = {

          getValue: () => (textarea ? textarea.value : ''),

          setValue: (v) => {

            if (textarea) textarea.value = v;

          },

          // minimal stubs used elsewhere

          on: () => {},

          setSize: () => {},

        };

        return;

      }



      codeEditor = CodeMirror.fromTextArea(textarea, {

        mode: 'javascript',

        theme: 'dracula',

        lineNumbers: true,

        autoCloseBrackets: true,

        matchBrackets: true,

        extraKeys: {

          'Ctrl-Space': 'autocomplete',

          'Ctrl-Enter': runCode,
```

```
    },
  });
})();


// Sensor Parameters Configuration
const sensorParameters = {
  Ultrasonic: {
    type: 'cone',
    description: 'Measures distance using ultrasonic sound waves',
    minRange: 0.1,
    maxRange: 10,
    defaultRange: 5,
    minFov: 5,
    maxFov: 60,
    defaultFov: 30,
    unit: 'm',
    color: '#00a8ff',
  },
  Radar: {
    type: 'cone',
    description: 'Radio detection and ranging system',
    minRange: 1,
    maxRange: 1000,
    defaultRange: 500,
    minFov: 1,
    maxFov: 45,
```

```
    defaultFov: 15,
    unit: 'm',
    color: '#ff9f1c',
  },
  LiDAR: {
    type: 'cone',
    description: 'Light detection and ranging system',
    minRange: 0.5,
    maxRange: 200,
    defaultRange: 100,
    minFov: 0.5,
    maxFov: 30,
    defaultFov: 10,
    unit: 'm',
    color: '#2ecc71',
  },
  Proximity: {
    type: 'sphere',
    description: 'Detects objects within a spherical range',
    minRange: 0.1,
    maxRange: 50,
    defaultRange: 10,
    unit: 'm',
    color: '#9b59b6',
  },
  Temperature: {
```

```
    type: 'point',
    description: 'Measures ambient temperature',
    color: '#e74c3c',
  },
  Humidity: {
    type: 'point',
    description: 'Measures relative humidity',
    color: '#3498db',
  },
  Pressure: {
    type: 'point',
    description: 'Measures atmospheric pressure',
    color: '#1abc9c',
  },
};

// Core Functions ============================================

function runCode() {
  try {
    cleanupEventHandlers();
    clearSceneEntities();

    const code = codeEditor.getValue();
    const safeCode = sanitizeCode(code);
```

```
      const func = new Function('viewer', 'Cesium', 'console', safeCode);

      const customConsole = {
        log: (...args) => outputToConsole(args.join(' '), 'log'),
        error: (...args) => outputToConsole(args.join(' '), 'error'),
        warn: (...args) => outputToConsole(args.join(' '), 'warning'),
      };

      func(viewer, Cesium, customConsole);
      outputToConsole('Code executed successfully', 'success');
    } catch (error) {
      outputToConsole(`Execution error: ${error.message}`, 'error');
      console.error(error);
    }
  }

  function sanitizeCode(code) {
    const forbiddenPatterns = [
      /eval\(/gi,
      /new Function\(/gi,
      /document\./gi,
      /window\./gi,
      /localStorage/gi,
      /XMLHttpRequest/gi,
      /fetch\(/gi,
      /\.innerHTML/gi,
```

```
      /import\(/gi,
      /require\(/gi,
    ];

    forbiddenPatterns.forEach((pattern) => {
      if (pattern.test(code)) {
        throw new Error('Code contains forbidden patterns');
      }
    });

    return code;
  }

  function cleanupEventHandlers() {
    state.activeHandlers.forEach((handler) => {
      if (handler?.destroy) handler.destroy();
    });
    state.activeHandlers = [];
  }

  function clearSceneEntities() {
    viewer.entities.removeAll();
    state.entities = [];
  }

  // UI Functions ================================================
```

```
    function initializeUI() {
      // Tab functionality
      document.querySelectorAll('.sidebar-tab').forEach((tab) => {
        tab.addEventListener('click', () => {
          document.querySelectorAll('.sidebar-tab').forEach((t) =>
t.classList.remove('active'));
          document.querySelectorAll('.tab-content').forEach((c) =>
c.classList.remove('active'));

          tab.classList.add('active');

document.getElementById(`${tab.dataset.tab}-tab`).classList.add('active');
        });
      });

      // File explorer
      document.querySelectorAll('.file-item').forEach((item) => {
        item.addEventListener('click', () =>
loadFileContent(item.dataset.file));
      });

      // Buttons
      document.getElementById('runButton').addEventListener('click', runCode);
      document.getElementById('stopButton').addEventListener('click',
stopExecution);
      document
        .getElementById('saveButton')
```

```
        .addEventListener('click', () => outputToConsole('Code saved',
'info'));

      document.getElementById('clearConsole').addEventListener('click', () =>
{

        document.getElementById('consoleOutput').textContent = '';

      });



      // Upload modal

      setupUploadModal();



      // Properties panel

      setupPropertiesPanel();



      // Toggle sidebar

      document.getElementById('toggleSidebar').addEventListener('click',
toggleSidebar);



      // AI Assistant

      setupAIAssistant();



      // Entity selection

      setupEntitySelection();



      // Sensor drag and drop

      setupSensorDragDrop();

    }
```

```
      function loadFileContent(filename) {
        // persist current buffer
        const current = state.currentFile || 'main.js';
        state.files[current] = codeEditor.getValue();

        state.currentFile = filename;
        document.getElementById('editorTitle').textContent = filename;

        // restore existing file if present
        if (state.files[filename]) {
          codeEditor.setValue(state.files[filename]);
          return;
        }

// fallback templates only if never loaded before
if (filename === 'scene.js') {
 const defaultSceneCode = `// Scene configuration file

function configureScene() {
 // Set lighting
 viewer.scene.globe.enableLighting = true;
 viewer.scene.shadowMap.enabled = true;

 // Set atmosphere effects
 viewer.scene.skyAtmosphere.show = true;
 viewer.scene.fog.enabled = true;
```

```
 console.log("Scene configured");
}

// Apply configuration
configureScene();`;

 state.files[filename] = defaultSceneCode;
 codeEditor.setValue(defaultSceneCode);

} else if (filename === 'main.js') {
 codeEditor.setValue(`// Digital Twin Scripting Environment
// Access the Cesium viewer with the 'viewer' variable

// Example: Add a simple point to the scene
function addPoint(lon, lat, height) {
 return viewer.entities.add({
   position: Cesium.Cartesian3.fromDegrees(lon, lat, height),
   point: {
     pixelSize: 10,
     color: Cesium.Color.RED,
     outlineColor: Cesium.Color.WHITE,
     outlineWidth: 2
   }
 });
}
```

```
// Example: Add a 3D model
function addModel(lon, lat, height, modelUrl) {
 return viewer.entities.add({
   position: Cesium.Cartesian3.fromDegrees(lon, lat, height),
   model: {
     uri: modelUrl,
     minimumPixelSize: 128,
     maximumScale: 20000
   }
 });
}

// Example: Fly to a location
function flyTo(lon, lat, height = 1000) {
 viewer.camera.flyTo({
   destination: Cesium.Cartesian3.fromDegrees(lon, lat, height),
   orientation: {
     heading: Cesium.Math.toRadians(0.0),
     pitch: Cesium.Math.toRadians(-45.0),
     roll: 0.0
   }
 });
}

// Initialize the scene
```

```
function initScene() {
 // Set initial camera position
 flyTo(-74.0060, 40.7128, 5000);

 // Add a sample point
 addPoint(-74.0060, 40.7128, 100);

 console.log("Digital Twin environment initialized");
}

initScene();`);
      } else if (state.uploadedModels[filename]) {
        const model = state.uploadedModels[filename];
        const pos = model.entity.position.getValue(viewer.clock.currentTime);
        const cartographic = Cesium.Cartographic.fromCartesian(pos);
        const lon = Cesium.Math.toDegrees(cartographic.longitude);
        const lat = Cesium.Math.toDegrees(cartographic.latitude);
        const height = cartographic.height;

        codeEditor.setValue(`// ${filename} Model Configuration
function configureModel() {
 const model = viewer.entities.add({
   name: "${filename}",
   position: Cesium.Cartesian3.fromDegrees(${lon}, ${lat}, ${height}),
   model: {
     uri: "${model.url}",
```

```
    minimumPixelSize: 128,
    maximumScale: 20000
   }
 });

 console.log("Model '${filename}' configured");
}

configureModel();`);
      }
    }

    function stopExecution() {
      if (viewer.camera.currentFlight) {
        viewer.camera.currentFlight.cancel();
      }
      clearSceneEntities();
      cleanupEventHandlers();
      outputToConsole('Execution stopped', 'info');
    }

    // Entity Management ============================================

    function makeEntityDraggable(entity) {
      let isDragging = false;
      const handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
```

```
handler.setInputAction((click) => {
  const pickedObject = viewer.scene.pick(click.position);
  if (Cesium.defined(pickedObject) && pickedObject.id === entity) {
    isDragging = true;
    viewer.scene.screenSpaceCameraController.enableRotate = false;
    viewer.canvas.style.cursor = 'move';
  }
}, Cesium.ScreenSpaceEventType.LEFT_DOWN);


handler.setInputAction((movement) => {
  if (isDragging) {
    const ray = viewer.camera.getPickRay(movement.endPosition);
    const position = viewer.scene.globe.pick(ray, viewer.scene);
    if (position) {
      entity.position = position;
      updatePropertiesPanelIfOpen(entity);
    }
  }
}, Cesium.ScreenSpaceEventType.MOUSE_MOVE);


handler.setInputAction(() => {
  if (isDragging) {
    isDragging = false;
    viewer.scene.screenSpaceCameraController.enableRotate = true;
    viewer.canvas.style.cursor = 'default';
```

```
        }

      }, Cesium.ScreenSpaceEventType.LEFT_UP);



      state.activeHandlers.push(handler);

    }



    function updatePropertiesPanelIfOpen(entity) {

      if (state.selectedEntity === entity) {

        const cartographic =
Cesium.Cartographic.fromCartesian(entity.position);

        document.getElementById('entityLon').value = Cesium.Math.toDegrees(

          cartographic.longitude

        ).toFixed(6);

        document.getElementById('entityLat').value = Cesium.Math.toDegrees(

          cartographic.latitude

        ).toFixed(6);

        document.getElementById('entityHeight').value =
cartographic.height.toFixed(2);

      }

    }



    // Upload Modal ================================================



    function setupUploadModal() {

      document.getElementById('uploadModelBtn').addEventListener('click', ()
=> {

        document.getElementById('uploadModal').style.display = 'flex';
```

```
      });

      document.getElementById('cancelUpload').addEventListener('click', () =>
{
        document.getElementById('uploadModal').style.display = 'none';
      });

      document.getElementById('selectModelFile').addEventListener('click', ()
=> {
        document.getElementById('modelFile').click();
      });

      document.getElementById('modelFile').addEventListener('change', (e) => {
        document.getElementById('selectedFileName').textContent =
          e.target.files[0]?.name || 'No file selected';
      });

      document.getElementById('uploadForm').addEventListener('submit',
handleModelUpload);
    }

    // UPDATED: keep local behavior and add optional server persistence
    async function handleModelUpload(e) {
      e.preventDefault();

      const modelName = document.getElementById('modelName').value;
      const lon = parseFloat(document.getElementById('modelLon').value);
```

```
      const lat = parseFloat(document.getElementById('modelLat').value);
      const height = parseFloat(document.getElementById('modelHeight').value);
      const modelFile = document.getElementById('modelFile').files[0];
      const persistOnServerEl = document.getElementById('persistOnServer');
      const persistOnServer = persistOnServerEl ? persistOnServerEl.checked :
false;

      if (!modelFile) {
        outputToConsole('Error: No model file selected', 'error');
        return;
      }

      try {
        outputToConsole(`Uploading model into scene: ${modelName}...`,
'info');

        // 1) CURRENT BEHAVIOR: local Blob URL + Cesium entity
        const modelUrl = URL.createObjectURL(modelFile);
        const entity = viewer.entities.add({
          name: modelName,
          position: Cesium.Cartesian3.fromDegrees(lon, lat, height),
          model: {
            uri: modelUrl,
            minimumPixelSize: 128,
            maximumScale: 20000
          },
          draggable: true
```

```
        });

        makeEntityDraggable(entity);
        state.entities.push(entity);
        state.uploadedModels[modelName] = { url: modelUrl, entity };

        updateEntitiesList();
        addModelToFileList(modelName, modelUrl);

        outputToConsole(
          `Added model to scene: ${modelName} at (${lon}, ${lat}, ${height})`,
          'success'
        );

        // 2) NEW BEHAVIOR: optional persistence to Flask backend
        if (persistOnServer) {
          outputToConsole('Persisting model on server (/upload-model)...',
'info');

          try {
            const formData = new FormData();
            // 'model' matches the expected field name in the Flask endpoint
            formData.append('model', modelFile, modelFile.name);

            const resp = await fetch('/upload-model', {
              method: 'POST',
```

```
              body: formData

            });



            if (!resp.ok) {

              const text = await resp.text();

              throw new Error(text || `HTTP ${resp.status}`);

            }



            outputToConsole('Model successfully saved on server. Browse at
/models.', 'success');

          } catch (serverErr) {

            outputToConsole(

              `Could not save model to server: ${serverErr.message}`,

              'warning'

            );

          }

        }



        // 3) Clean up UI and fly to model

        document.getElementById('uploadModal').style.display = 'none';

        document.getElementById('uploadForm').reset();

        document.getElementById('selectedFileName').textContent = 'No file
selected';



        viewer.flyTo(entity);

      } catch (error) {

        outputToConsole(`Error uploading model: ${error.message}`, 'error');
```

```
        console.error(error);
      }
    }

    function addModelToFileList(name, url) {
      const fileExplorer = document.querySelector('.file-explorer');
      const newItem = document.createElement('li');
      newItem.className = 'file-item';
      newItem.dataset.file = name;
      newItem.innerHTML = `<i class="fas fa-cube"></i> ${name}`;
      newItem.addEventListener('click', () => loadFileContent(name));
      fileExplorer.appendChild(newItem);
    }

    // Properties Panel ============================================

    function setupPropertiesPanel() {

document.getElementById('closePropertiesPanel').addEventListener('click', () =>
{
        document.getElementById('propertiesPanel').style.display = 'none';
        state.selectedEntity = null;
      });

      document.getElementById('updateEntity').addEventListener('click',
updateEntity);
      document.getElementById('deleteEntity').addEventListener('click',
deleteEntity);
```

```
    }

    function showPropertiesPanel(entity) {
      state.selectedEntity = entity;
      const cartographic = Cesium.Cartographic.fromCartesian(
        entity.position.getValue(viewer.clock.currentTime)
      );

      document.getElementById('entityName').value = entity.name || '';
      document.getElementById('entityLon').value = Cesium.Math.toDegrees(
        cartographic.longitude
      ).toFixed(6);
      document.getElementById('entityLat').value = Cesium.Math.toDegrees(
        cartographic.latitude
      ).toFixed(6);
      document.getElementById('entityHeight').value =
cartographic.height.toFixed(2);
      document.getElementById('propertiesPanel').style.display = 'block';
    }

    function updateEntity() {
      if (!state.selectedEntity) return;

      const name = document.getElementById('entityName').value;
      const lon = parseFloat(document.getElementById('entityLon').value);
      const lat = parseFloat(document.getElementById('entityLat').value);
```

```
      const height =
parseFloat(document.getElementById('entityHeight').value);



      state.selectedEntity.name = name;

      state.selectedEntity.position = Cesium.Cartesian3.fromDegrees(lon, lat,
height);



      if (state.uploadedModels[name]) {

        state.uploadedModels[name].entity = state.selectedEntity;

      }



      outputToConsole(`Updated entity: ${name}`, 'success');

      updateEntitiesList();

    }



    function deleteEntity() {

      if (!state.selectedEntity) return;



      const name = state.selectedEntity.name || 'Unnamed entity';

      viewer.entities.remove(state.selectedEntity);

      state.entities = state.entities.filter((e) => e !==
state.selectedEntity);



      if (state.uploadedModels[name]) {

        URL.revokeObjectURL(state.uploadedModels[name].url);

        delete state.uploadedModels[name];

      }
```

```
      state.selectedEntity = null;
      document.getElementById('propertiesPanel').style.display = 'none';
      outputToConsole(`Deleted entity: ${name}`, 'success');
      updateEntitiesList();
    }

    function updateEntitiesList() {
      const entitiesList = document.getElementById('entities-list');
      entitiesList.innerHTML = '';

      state.entities.forEach((entity) => {
        const item = document.createElement('div');
        item.className = 'file-item';
        item.innerHTML = `<i class="fas fa-cube"></i> ${entity.name ||
'Unnamed entity'}`;
        item.addEventListener('click', () => {
          viewer.flyTo(entity);
          showPropertiesPanel(entity);
        });
        entitiesList.appendChild(item);
      });
    }

    // UI Helpers =================================================
```

```
    function toggleSidebar() {
      const sidebar = document.getElementById('sidebar');
      sidebar.classList.toggle('collapsed');

      const icon =
document.getElementById('toggleSidebar').querySelector('i');
      icon.classList.toggle('fa-chevron-left');
      icon.classList.toggle('fa-chevron-right');
    }

    function outputToConsole(message, type = 'log') {
      const consoleOutput = document.getElementById('consoleOutput');
      const messageDiv = document.createElement('div');
      messageDiv.className = `log-${type}`;
      messageDiv.textContent = message;
      consoleOutput.appendChild(messageDiv);
      consoleOutput.scrollTop = consoleOutput.scrollHeight;
    }

    // Entity Selection ========================================

    function setupEntitySelection() {
      const handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
      state.activeHandlers.push(handler);

      handler.setInputAction((movement) => {
```

```
        const pickedObject = viewer.scene.pick(movement.endPosition);
        if (Cesium.defined(pickedObject) && pickedObject.id) {
          viewer.canvas.style.cursor = 'pointer';
          showTooltip(movement.endPosition, pickedObject.id.name || 'Click to
inspect');
        } else {
          viewer.canvas.style.cursor = 'default';
          hideTooltip();
        }
      }, Cesium.ScreenSpaceEventType.MOUSE_MOVE);

      handler.setInputAction((click) => {
        const pickedObject = viewer.scene.pick(click.position);
        if (Cesium.defined(pickedObject) && pickedObject.id) {
          showPropertiesPanel(pickedObject.id);
        }
      }, Cesium.ScreenSpaceEventType.LEFT_CLICK);
    }

    function showTooltip(position, message) {
      const tooltip = document.getElementById('tooltip');
      tooltip.style.left = `${position.x + 10}px`;
      tooltip.style.top = `${position.y + 10}px`;
      tooltip.textContent = message;
      tooltip.style.display = 'block';
    }
```

```
    function hideTooltip() {
      document.getElementById('tooltip').style.display = 'none';
    }

    // Sensor Drag and Drop ====================================

    function setupSensorDragDrop() {
      document.querySelectorAll('.sensor-item').forEach((item) => {
        item.addEventListener('dragstart', (e) => {
          const sensorType = item.getAttribute('data-sensor-type');
          const sensorShape = item.getAttribute('data-sensor-shape');
          const sensorParam = { ...sensorParameters[sensorType], shape:
sensorShape };
          e.dataTransfer.setData('application/json',
JSON.stringify(sensorParam));
          e.dataTransfer.effectAllowed = 'copy';
        });
      });

      viewer.scene.canvas.addEventListener('dragover', (e) => {
        e.preventDefault();
        e.dataTransfer.dropEffect = 'copy';
      });

      viewer.scene.canvas.addEventListener('drop', (e) => {
        e.preventDefault();
```

```
        try {
          const sensorParam =
JSON.parse(e.dataTransfer.getData('application/json'));
          const pickPosition = new Cesium.Cartesian2(e.offsetX, e.offsetY);
          const earthPosition = viewer.scene.pickPosition(pickPosition);

          if (Cesium.defined(earthPosition)) {
            createSensorEntity(sensorParam, earthPosition);
            outputToConsole(`Added ${sensorParam.type} sensor to scene`,
'success');
          }
        } catch (error) {
          outputToConsole(`Error adding sensor: ${error.message}`, 'error');
        }
      });
    }

    function createSensorEntity(sensorParam, position) {
      const entityOptions = {
        position: position,
        name: `${sensorParam.type} Sensor`,
        description: sensorParam.description,
        label: {
          text: sensorParam.type,
          font: '14pt sans-serif',
          style: Cesium.LabelStyle.FILL_AND_OUTLINE,
          fillColor: Cesium.Color.WHITE,
```

```
          outlineColor: Cesium.Color.BLACK,
          outlineWidth: 2,
          verticalOrigin: Cesium.VerticalOrigin.BOTTOM,
          pixelOffset: new Cesium.Cartesian2(0, -20),
          disableDepthTestDistance: Number.POSITIVE_INFINITY,
        },
      };

      switch (sensorParam.shape) {
        case 'cone':
          entityOptions.cylinder = {
            length: sensorParam.defaultRange,
            topRadius: 0,
            bottomRadius:
              sensorParam.defaultRange *
              Math.tan(Cesium.Math.toRadians(sensorParam.defaultFov / 2)),
            material:
Cesium.Color.fromCssColorString(sensorParam.color).withAlpha(0.4),
            outline: true,
            outlineColor: Cesium.Color.WHITE,
            slices: 64,
          };
          break;
        case 'sphere':
          entityOptions.ellipsoid = {
            radii: new Cesium.Cartesian3(
```

```
              sensorParam.defaultRange,
              sensorParam.defaultRange,
              sensorParam.defaultRange
            ),
            material:
Cesium.Color.fromCssColorString(sensorParam.color).withAlpha(0.3),
            outline: true,
            outlineColor: Cesium.Color.WHITE,
          };
          break;
        case 'point':
          entityOptions.point = {
            pixelSize: 10,
            color: Cesium.Color.fromCssColorString(sensorParam.color),
            outlineColor: Cesium.Color.WHITE,
            outlineWidth: 2,
          };
          break;
      }

      const entity = viewer.entities.add(entityOptions);
      state.entities.push(entity);
      updateEntitiesList();
      return entity;
    }
```

```
    // AI Assistant ================================================



    function setupAIAssistant() {

      document.getElementById('aiAssistantButton').addEventListener('click',
() => {

        document.getElementById('aiAssistantPanel').style.display = 'block';

        document.getElementById('aiInput').focus();

      });




document.getElementById('closeAIAssistantPanel').addEventListener('click', ()
=> {

        document.getElementById('aiAssistantPanel').style.display = 'none';

      });



      document.getElementById('aiSend').addEventListener('click',
processAIQuery);

      document.getElementById('aiInput').addEventListener('keypress', (e) => {

        if (e.key === 'Enter') processAIQuery();

      });

    }



      function collectIDEContext() {

        // Ensure file registry exists

        state.files = state.files || {};



        // Persist current editor buffer into the active file

        const activeFile = state.currentFile || 'main.js';
```

```
    state.files[activeFile] = codeEditor.getValue();


    // OPTIONAL: lightweight scene snapshot (safe, non-heavy)
    const sceneSnapshot = {
      entityCount: viewer.entities.values.length,
      entities: viewer.entities.values.slice(0, 20).map(e => ({
        name: e.name || 'unnamed',
        hasModel: !!e.model,
        hasPoint: !!e.point,
        hasLabel: !!e.label
      }))
    };


    return {
      current_file: activeFile,
      files: { ...state.files },
      scene: sceneSnapshot
    };
  }



  function applyUnifiedDiff(original, diffText) {
    try {
      const origLines = original.split('\n');
      const diffLines = diffText.split('\n');
```

```
          let out = [];
          let i = 0;
          let d = 0;

          while (d < diffLines.length) {
            const line = diffLines[d];

            if (line.startsWith('@@')) {
              const match = line.match(/@@ -(\d+)/);
              const target = parseInt(match[1], 10) - 1;
              out.push(...origLines.slice(i, target));
              i = target;
              d++;
              continue;
            }

            if (line.startsWith('+')) {
              out.push(line.slice(1));
            } else if (line.startsWith('-')) {
              i++;
            } else if (line.startsWith(' ')) {
              out.push(origLines[i]);
              i++;
            }

            d++;
```

```
      }

      out.push(...origLines.slice(i));

      return { ok: true, text: out.join('\n') };
    } catch (e) {
      return { ok: false, error: e.message };
    }
  }

  function applyAIPatches(patches) {
    if (!Array.isArray(patches)) {
      throw new Error('Invalid patches format');
    }

    patches.forEach(patch => {
      const { file, op, diff } = patch;

      if (op !== 'patch') {
        throw new Error(`Unsupported patch op: ${op}`);
      }

      if (!state.files[file]) {
        throw new Error(`Unknown file: ${file}`);
      }
```

```
      const original = state.files[file];
      const result = applyUnifiedDiff(original, diff);

      if (!result.ok) {
        throw new Error(`Patch failed for ${file}: ${result.error}`);
      }

      state.files[file] = result.text;

      // Update editor if this is the active file
      if (state.currentFile === file) {
        codeEditor.setValue(result.text);
      }
    });
  }
  async function processAIQuery() {
    const input = document.getElementById('aiInput');
    const query = input.value.trim();
    if (!query) return;

    // Show user message
    addAIMessage(query, 'user');
    input.value = '';

    try {
      // Handle built-in commands first (fly to, clear scene, etc.)
```

```
        if (await handleSpecialCommands(query)) return;

        // Show thinking indicator
        addAIMessage('Thinking...', 'assistant');

        // Ensure current editor buffer is persisted
        const currentFile = state.currentFile || 'main.js';
        state.files = state.files || {};
        state.files[currentFile] = codeEditor.getValue();

        // Call YOUR BACKEND (never OpenAI directly)
        const response = await fetch('/ai_query', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({
            user_prompt: query,
            ide: collectIDEContext()
          })
        });

        if (!response.ok) {
          const text = await response.text();
          throw new Error(text || `Server error ${response.status}`);
        }

        const data = await response.json();
```

```
          // Remove "Thinking..."
          const contentDiv = document.getElementById('aiAssistantContent');
          if (contentDiv.lastChild?.textContent === 'Thinking...') {
            contentDiv.removeChild(contentDiv.lastChild);
          }

          if (!data.success) {
            throw new Error(data.error || 'AI request failed');
          }

          // Display AI explanation
          addAIMessage(data.message || 'AI response received.', 'assistant');

          // ---- SAFETY BLOCK ----
          if (data.safety?.blocked) {
            addAIMessage(
              '⚠️ Changes were blocked by safety validation and were NOT applied.',
              'assistant'
            );

            if (data.safety.errors?.length) {
              addAIMessage(
                `Errors:\n- ${data.safety.errors.join('\n- ')}`,
                'assistant'
```

```
              );
            }
            return;
          }

          // ---- APPLY PATCHES ----
          if (Array.isArray(data.patches) && data.patches.length > 0) {
            try {
              applyAIPatches(data.patches);
              addAIMessage('Patches successfully applied to IDE.',
'assistant');
            } catch (patchErr) {
              addAIMessage(`Patch application failed: ${patchErr.message}`,
'assistant');
              console.error(patchErr);
              return;
            }
          } else {
            addAIMessage('No code changes were required.', 'assistant');
          }

          // ---- WARNINGS ----
          if (data.safety?.warnings?.length) {
            addAIMessage(
              `Warnings:\n- ${data.safety.warnings.join('\n- ')}`,
              'assistant'
            );
```

```
        }

        // ---- FOCUS FILE ----
        if (data.suggested_actions?.focus_file) {
          const focusFile = data.suggested_actions.focus_file;
          if (state.files[focusFile]) {
            loadFileContent(focusFile);
          }
        }

        // ---- AUTO RUN ----
        if (data.suggested_actions?.run_after_apply) {
          runCode();
        }

      } catch (error) {
        // Remove "Thinking..." if still present
        const contentDiv = document.getElementById('aiAssistantContent');
        if (contentDiv.lastChild?.textContent === 'Thinking...') {
          contentDiv.removeChild(contentDiv.lastChild);
        }

        addAIMessage(`AI Error: ${error.message}`, 'assistant');
        console.error('AI Error:', error);

        addAIMessage(
```

```
            'Troubleshooting: Ensure backend is running, /ai_query exists, and
OPENAI_API_KEY is set server-side.',
            'assistant'
          );
        }
      }



    async function handleSpecialCommands(query) {
      const lowerQuery = query.toLowerCase();

      // Location commands
      if (
        lowerQuery.includes('show me') ||
        lowerQuery.includes('fly to') ||
        lowerQuery.includes('take me to')
      ) {
        await handleLocationQuery(query);
        return true;
      }

      // Clear commands
      if (lowerQuery.includes('clear scene')) {
        stopExecution();
        addAIMessage('Scene cleared', 'assistant');
        return true;
```

```
    }

    if (lowerQuery.includes('clear console')) {
      document.getElementById('consoleOutput').textContent = '';
      addAIMessage('Console cleared', 'assistant');
      return true;
    }

    // Help command
    if (lowerQuery.includes('help') || lowerQuery === '?') {
      showHelp();
      return true;
    }

    // Toggle code execution
    if (lowerQuery.includes('disable ai code')) {
      state.isAICodeExecutionEnabled = false;
      addAIMessage('AI code execution is now disabled', 'assistant');
      return true;
    }

    if (lowerQuery.includes('enable ai code')) {
      state.isAICodeExecutionEnabled = true;
      addAIMessage('AI code execution is now enabled', 'assistant');
      return true;
    }
```

```
      return false;

    }



    function showHelp() {

      addAIMessage(`I can help you with:`, 'assistant');

      addAIMessage(`1. Code Editing - "Add a rotating 3D model"`,
'assistant');

      addAIMessage(`2. Scene Control - "Create a grid of points"`,
'assistant');

      addAIMessage(`3. Simulations - "Simulate drone flight path"`,
'assistant');

      addAIMessage(`4. Location - "Show me the Eiffel Tower"`, 'assistant');

      addAIMessage(`5. Learning - "How to add animations?"`, 'assistant');

      addAIMessage(`Try asking me to modify the code or create simulations!`,
'assistant');

    }



    function handleAIResponse(message, originalQuery) {

      // Check for code blocks first

      const codeMatch = message.match(/```javascript([\s\S]*?)```/);



      if (codeMatch && codeMatch[1] && state.isAICodeExecutionEnabled) {

        const code = codeMatch[1].trim();

        const responseWithoutCode = message.replace(

          /```javascript[\s\S]*?```/,

          '[Code ready for review]'

        );
```

```
        addAIMessage(responseWithoutCode, 'assistant');


        // Create action buttons
        const actionDiv = document.createElement('div');
        actionDiv.className = 'ai-message ai-assistant';
        actionDiv.innerHTML = `
    <div class="ai-actions">
      <button class="btn btn-sm" id="replaceCode">
        <i class="fas fa-code"></i> Replace Entire Code
      </button>
      <button class="btn btn-sm" id="appendCode">
        <i class="fas fa-plus"></i> Append to Code
      </button>
      <button class="btn btn-sm" id="executeOnly">
        <i class="fas fa-play"></i> Execute Without Saving
      </button>
      <button class="btn btn-sm btn-outline" id="copyCode">
        <i class="fas fa-copy"></i> Copy to Clipboard
      </button>
    </div>
    <div class="code-preview" contenteditable="true">${escapeHtml(code)}</div>
  `;


        document.getElementById('aiAssistantContent').appendChild(actionDiv);
```

```
        // Set up button handlers
        document.getElementById('replaceCode').addEventListener('click', () =>
{
          codeEditor.setValue(code);
          addAIMessage('Code replaced in editor', 'assistant');
          actionDiv.remove();
        });

        document.getElementById('appendCode').addEventListener('click', () =>
{
          codeEditor.setValue(codeEditor.getValue() + '\n\n' + code);
          addAIMessage('Code appended to editor', 'assistant');
          actionDiv.remove();
        });

        document.getElementById('executeOnly').addEventListener('click', () =>
{
          const originalCode = codeEditor.getValue();
          codeEditor.setValue(code);
          runCode();
          codeEditor.setValue(originalCode);
          addAIMessage('Code executed without saving', 'assistant');
          actionDiv.remove();
        });

        document.getElementById('copyCode').addEventListener('click', () => {
          navigator.clipboard.writeText(code);
```

```
          addAIMessage('Code copied to clipboard', 'assistant');
          actionDiv.remove();
        });
      } else {
        // No code found, just show the message
        addAIMessage(message, 'assistant');

        // If the query was asking for code but didn't get any
        if (
          originalQuery.toLowerCase().includes('code') ||
          originalQuery.toLowerCase().includes('how to') ||
          originalQuery.toLowerCase().includes('example')
        ) {
          addAIMessage('Would you like me to generate example code for this?',
'assistant');

          const followUpDiv = document.createElement('div');
          followUpDiv.className = 'ai-message ai-assistant';
          followUpDiv.innerHTML = `
      <button class="btn btn-sm" id="generateExample">
        <i class="fas fa-magic"></i> Yes, Generate Code
      </button>
    `;

document.getElementById('aiAssistantContent').appendChild(followUpDiv);
```

```
          document.getElementById('generateExample').addEventListener('click',
() => {

            processAIQuery(`Generate complete working code for:
${originalQuery}`);

            followUpDiv.remove();

          });

        }

      }

    }



    async function handleLocationQuery(query) {

      const locationMatch = query.match(/show me (.*)|fly to (.*)|take me to
(.*)/i);

      const location = locationMatch?.[1] || locationMatch?.[2] ||
locationMatch?.[3];



      if (!location) {

        addAIMessage("Please specify a location (e.g., 'Show me Paris')",
'assistant');

        return;

      }



      addAIMessage(`Searching for ${location}...`, 'assistant');



      try {

        const response = await fetch(


`https://nominatim.openstreetmap.org/search?format=json&q=${encodeURIComponent(

            location
```

```
          )}`
        );
        const data = await response.json();

        if (data.length > 0) {
          const { lat, lon, display_name } = data[0];
          viewer.camera.flyTo({
            destination: Cesium.Cartesian3.fromDegrees(parseFloat(lon),
parseFloat(lat), 2000),
            orientation: {
              heading: Cesium.Math.toRadians(0.0),
              pitch: Cesium.Math.toRadians(-45.0),
              roll: 0.0,
            },
          });
          addAIMessage(`Found: ${display_name} (${lat}, ${lon})`,
'assistant');

          // Offer to add a marker
          const markerDiv = document.createElement('div');
          markerDiv.className = 'ai-message ai-assistant';
          markerDiv.innerHTML = `
       <button class="btn btn-sm" id="addMarker">
         <i class="fas fa-map-marker-alt"></i> Add Marker at This Location
       </button>
     `;
```

```
document.getElementById('aiAssistantContent').appendChild(markerDiv);



          document.getElementById('addMarker').addEventListener('click', () =>
{

            const markerCode = `// Marker at ${display_name}

addPoint(${lon}, ${lat}, 100, Cesium.Color.RED);`;

            codeEditor.setValue(codeEditor.getValue() + '\n\n' + markerCode);

            runCode();

            markerDiv.remove();

          });

        } else {

          addAIMessage(`Location not found: ${location}`, 'assistant');

        }

      } catch (error) {

        addAIMessage(`Geocoding error: ${error.message}`, 'assistant');

      }

    }



    function addAIMessage(message, sender) {

      const contentDiv = document.getElementById('aiAssistantContent');

      const messageDiv = document.createElement('div');

      messageDiv.className = `ai-message ai-${sender}`;

      messageDiv.textContent = message;

      contentDiv.appendChild(messageDiv);

      contentDiv.scrollTop = contentDiv.scrollHeight;

    }
```

```
    // Helper function to escape HTML
    function escapeHtml(unsafe) {
      return unsafe
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;')
        .replace(/"/g, '&quot;')
        .replace(/'/g, '&#039;');
    }
    // Initialization ==============================================


    function initialize() {
      initializeScene();
      initializeUI();
      setTimeout(runCode, 500);
    }


    // Start the application
    initialize();
  </script>
 </body>
</html>
```

Server_Host.py

```python
# Server_Host.py

"""

Flask server for the OPENQQUANTIFY Digital Twin IDE.



Responsibilities:

- Serve the IDE frontend

- Accept AI queries WITH full IDE context (multi-file, active file, scene
snapshot)

- Build indexed prompts for OpenAI

- Return structured DIFF patches per file (not full replacements)

- Perform server-side validation (dry-run checks) before responding

- Enforce physics-aware simulation guards

"""



import os

import re

import json

import difflib

from typing import Dict, List, Tuple, Any, Optional



from flask import Flask, render_template, request, jsonify,
send_from_directory, abort

from dotenv import load_dotenv

from openai import OpenAI



import routes.objects as objects_router
```

```
# ----------------------------------------------------------------------

# ENV SETUP

# ----------------------------------------------------------------------


load_dotenv()


CESIUM_ION_TOKEN = os.getenv("CESIUM_ION_TOKEN", "")

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY", "")

OPENAI_MODEL = os.getenv("OPENAI_MODEL", "gpt-4.1")  # default, override via
env


if not OPENAI_API_KEY:

   raise RuntimeError("OPENAI_API_KEY is not set")


client = OpenAI(api_key=OPENAI_API_KEY)




# ----------------------------------------------------------------------

# APP SETUP

# ----------------------------------------------------------------------


app = Flask(__name__, static_folder="static", template_folder="templates")

app.register_blueprint(objects_router.objects_bp)
```

```
# ---------------------------------------------------------------------

# CONSTANTS / GUARDS

# ---------------------------------------------------------------------



FORBIDDEN_JS_PATTERNS = [

   r"\beval\s*\(",

   r"\bnew\s+Function\s*\(",

   r"\bFunction\s*\(",

   r"\bwindow\.",

   r"\bdocument\.",

   r"\blocalStorage\b",

   r"\bsessionStorage\b",

   r"\bXMLHttpRequest\b",

   r"\bfetch\s*\(",

   r"\bWebSocket\b",

   r"\bimport\s*\(",

   r"\brequire\s*\(",

   r"\.innerHTML\b",

]



# Physics / simulation guard heuristics

PHYSICS_RISKY_PATTERNS = [

   r"while\s*\(\s*true\s*\)",

   r"for\s*\(\s*;\s*;\s*\)",

   r"setInterval\s*\(\s*[^,]+,\s*(0|1|2|3|4|5|6|7|8|9)\s*\)",  # ultra-fast
interval
```

```
    r"setTimeout\s*\(\s*[^,]+,\s*(0|1|2|3|4|5|6|7|8|9)\s*\)",    # ultra-fast timeout
]

# Cesium clock multiplier guard (too high will melt browsers)
MAX_CLOCK_MULTIPLIER = 200.0

# Limit output size to avoid megabytes of patch text
MAX_PATCH_CHARS = 80_000

# ---------------------------------------------------------------------
# UTILS: IDE CONTEXT + INDEXING
# ---------------------------------------------------------------------

def index_functions(code: str) -> List[str]:
    """Extract function names for lightweight indexing."""
    return re.findall(r"function\s+([a-zA-Z0-9_]+)\s*\(", code)


def normalize_ide_payload(payload: dict) -> dict:
    """
    Expected frontend payload schema:

    {
      "user_prompt": "...",
      "ide": {
```

```
      "current_file": "main.js",
      "files": { "main.js": "...", "scene.js": "...", ... },
      "scene": { ... optional snapshot ... }
    }
}
"""
ide = payload.get("ide") or {}
files = ide.get("files") or {}

# Hard safety: only allow str->str
cleaned_files: Dict[str, str] = {}
for k, v in files.items():
    if isinstance(k, str) and isinstance(v, str):
        cleaned_files[k] = v

ide["files"] = cleaned_files
ide["current_file"] = str(ide.get("current_file") or "main.js")

scene = ide.get("scene")
if scene is None:
    ide["scene"] = {}
elif not isinstance(scene, dict):
    ide["scene"] = {}

payload["ide"] = ide
return payload
```

```
def build_ai_system_prompt(ide: dict) -> str:
    """
    Build deterministic system prompt.
    IMPORTANT: We instruct the model to output JSON ONLY with patch operations.
    """
    files: Dict[str, str] = ide.get("files", {})
    current_file: str = ide.get("current_file", "main.js")
    scene_state = ide.get("scene", {})

    function_index = {fname: index_functions(content) for fname, content in
files.items()}

    file_blocks = []
    for fname, content in files.items():
        file_blocks.append(f"\n### FILE: {fname}\n{content}")

    # JSON output contract (the model MUST follow)
    # patches: array of {file, op, diff} where diff is unified diff from
original->new for that file
    # message: short assistant text
    # safety: optional warnings
    return f"""
You are a senior CesiumJS simulation engineer working inside a live Digital
Twin IDE.
```

```
Your output MUST be a single JSON object and nothing else.

No markdown. No code blocks. No prose outside JSON.


You have these rules:

- Only produce patches to existing project files.

- Do not output full-file replacements unless absolutely necessary.

- Never use eval, Function, window, document, fetch, WebSocket, localStorage.

- Prefer existing helpers/functions when possible.

- Keep changes minimal and well-scoped.

- Any physics or time-based simulation must be safe for browsers:

 - No infinite loops.

 - No ultra-fast timers (< 16ms).

 - No Cesium clock multiplier > {MAX_CLOCK_MULTIPLIER}.

 - Prefer viewer.clock.onTick for animation with guardable logic.


ACTIVE FILE: {current_file}


AVAILABLE FILES:

{json.dumps(list(files.keys()), indent=2)}


FUNCTION INDEX:

{json.dumps(function_index, indent=2)}


SCENE STATE (may be empty):

{json.dumps(scene_state, indent=2)}
```

```
PROJECT CODE:

{''.join(file_blocks)}



OUTPUT JSON SCHEMA (strict):

{{

 "message": "short helpful explanation of what you changed",

 "patches": [

   {{

     "file": "main.js",

     "op": "patch",

     "diff": "unified diff text that applies cleanly to the current file
content"

   }}

 ],

 "suggested_actions": {{

   "run_after_apply": true|false,

   "focus_file": "main.js"

 }},

 "safety": {{

   "warnings": ["..."],

   "blocked": false

 }}

}}

""".strip()
```

```
# ---------------------------------------------------------------------

# UTILS: VALIDATION / DRY-RUN CHECKS

# ---------------------------------------------------------------------



def scan_forbidden_patterns(text: str) -> List[str]:

   hits = []

   for pat in FORBIDDEN_JS_PATTERNS:

       if re.search(pat, text, flags=re.IGNORECASE | re.MULTILINE):

           hits.append(pat)

   return hits




def scan_physics_risks(text: str) -> List[str]:

   hits = []

   for pat in PHYSICS_RISKY_PATTERNS:

       if re.search(pat, text, flags=re.IGNORECASE | re.MULTILINE):

           hits.append(pat)



   # clock.multiplier guard (heuristic)

   # catches: viewer.clock.multiplier = 10000;

   m = re.findall(r"clock\.multiplier\s*=\s*([0-9]+(?:\.[0-9]+)?)", text,
flags=re.IGNORECASE)

   for val in m:

       try:

           f = float(val)

           if f > MAX_CLOCK_MULTIPLIER:
```

```
                hits.append(f"clock.multiplier too high: {f} >
{MAX_CLOCK_MULTIPLIER}")
        except Exception:
            pass

    return hits


def apply_unified_diff(original: str, unified_diff_text: str) -> Tuple[bool,
str]:
    """
    Apply a unified diff to `original` using a simple strategy:
    - We use difflib.restore style is not for unified diffs.
    - Instead, we implement a pragmatic patcher:
      - Parse unified diff hunks and apply line-based edits.
    This is not a full GNU patch implementation, but works well for
model-generated diffs
    when they match exact context.

    Returns: (ok, new_text_or_error)
    """
    try:
        diff_lines = unified_diff_text.splitlines()
        # Remove possible leading/trailing noise
        # Must contain --- and +++ and @@ hunks
        if not any(l.startswith("@@") for l in diff_lines):
            return False, "Diff missing hunks (@@ ... @@)."
```

```
        orig_lines = original.splitlines()

        out_lines = []

        i = 0  # index into orig_lines


        # We locate hunks and apply

        h = 0

        while h < len(diff_lines):

            line = diff_lines[h]

            if line.startswith("---") or line.startswith("+++"):

                h += 1

                continue


            if not line.startswith("@@"):

                h += 1

                continue


            # Parse hunk header: @@ -l,s +l,s @@

            m = re.match(r"@@\s*-(\d+)(?:,(\d+))?\s+\+(\d+)(?:,(\d+))?\s*@@",
line)

            if not m:

                return False, f"Invalid hunk header: {line}"


            old_start = int(m.group(1))

            # old_count = int(m.group(2) or "1")

            # new_start = int(m.group(3))
```

```
            # new_count = int(m.group(4) or "1")

            # Copy unchanged lines up to hunk start (1-based in diff)
            target_i = old_start - 1
            if target_i < i:
                # overlapping hunks or inconsistent diff
                return False, "Overlapping or inconsistent hunk positions."
            out_lines.extend(orig_lines[i:target_i])
            i = target_i

            # Apply hunk lines until next hunk header or EOF
            h += 1
            while h < len(diff_lines) and not diff_lines[h].startswith("@@"):
                dl = diff_lines[h]
                if dl.startswith("---") or dl.startswith("+++"):
                    h += 1
                    continue
                if dl.startswith(" "):
                    # context line must match
                    ctx = dl[1:]
                    if i >= len(orig_lines) or orig_lines[i] != ctx:
                        return False, f"Context mismatch applying diff at line
{i+1}."
                    out_lines.append(orig_lines[i])
                    i += 1
                elif dl.startswith("-"):
```

```
                    # removal line must match
                    rem = dl[1:]
                    if i >= len(orig_lines) or orig_lines[i] != rem:
                        return False, f"Removal mismatch applying diff at line
{i+1}."
                    i += 1
                elif dl.startswith("+"):
                    out_lines.append(dl[1:])
                else:
                    # unknown line type
                    return False, f"Unknown diff line: {dl}"
                h += 1

        # Copy remaining lines
        out_lines.extend(orig_lines[i:])
        return True, "\n".join(out_lines) + ("\n" if original.endswith("\n")
else "")

    except Exception as e:
        return False, f"Exception applying diff: {str(e)}"



def validate_and_simulate_patchset(files: Dict[str, str], patches: List[dict])
-> Tuple[bool, Dict[str, str], List[str], List[str]]:
    """
    Dry-run validation:
    - Ensure patches refer to known files
```

```
  - Apply patches to produce new_files

  - Scan for forbidden patterns and physics risks on the patched outputs

  Returns:

    ok, new_files, warnings, errors

  """

  warnings: List[str] = []

  errors: List[str] = []



  new_files = dict(files)



  for p in patches:

      fname = p.get("file")

      op = p.get("op")

      diff_text = p.get("diff")



      if not isinstance(fname, str) or not fname:

          errors.append("Patch missing valid 'file'.")

          continue

      if fname not in new_files:

          errors.append(f"Patch refers to unknown file: {fname}")

          continue

      if op != "patch":

          errors.append(f"Unsupported patch op '{op}' for file {fname}. Only
'patch' allowed.")

          continue

      if not isinstance(diff_text, str) or not diff_text.strip():
```

```
            errors.append(f"Patch for {fname} missing diff text.")
            continue

        ok, patched_or_err = apply_unified_diff(new_files[fname], diff_text)
        if not ok:
            errors.append(f"Failed to apply patch for {fname}:
{patched_or_err}")
            continue

        # Apply
        new_files[fname] = patched_or_err

        # Guard scans
        forbidden = scan_forbidden_patterns(new_files[fname])
        if forbidden:
            errors.append(f"Forbidden patterns in patched {fname}: {forbidden}")

        physics_risks = scan_physics_risks(new_files[fname])
        if physics_risks:
            # Not always fatal, but some are severe
            warnings.append(f"Physics/simulation risks in {fname}:
{physics_risks}")

    ok_final = (len(errors) == 0)

    # Also scan concatenated project for forbidden patterns (cross-file)
    joined = "\n\n".join(new_files.values())
```

```
    forbidden_all = scan_forbidden_patterns(joined)

    if forbidden_all:

        ok_final = False

        errors.append(f"Forbidden patterns found in project after patch:
{forbidden_all}")



    return ok_final, new_files, warnings, errors




def safe_json_loads(text: str) -> Tuple[bool, Any, str]:

    try:

        obj = json.loads(text)

        return True, obj, ""

    except Exception as e:

        return False, None, str(e)




def compute_unified_diff(original: str, updated: str, filename: str) -> str:

    """

    Helper: If you ever decide to fallback to full content, we can compute
diffs ourselves.

    Not used directly for model output, but can be useful if you switch
strategies.

    """

    orig_lines = original.splitlines(keepends=False)

    upd_lines = updated.splitlines(keepends=False)

    diff = difflib.unified_diff(
```

```
        orig_lines,
        upd_lines,
        fromfile=f"a/{filename}",
        tofile=f"b/{filename}",
        lineterm=""
    )
    return "\n".join(diff)



# ---------------------------------------------------------------------
# ROUTES
# ---------------------------------------------------------------------

@app.route("/")
def index():
    return render_template("digital_twin.html",
cesium_ion_token=CESIUM_ION_TOKEN)



@app.route("/ai_query", methods=["POST"])
def ai_query():
    payload = request.get_json(silent=True)
    if not payload:
        return jsonify({"success": False, "error": "Invalid JSON payload"}), 400

    payload = normalize_ide_payload(payload)
```

```
    user_prompt = payload.get("user_prompt")
    if not isinstance(user_prompt, str) or not user_prompt.strip():
        return jsonify({"success": False, "error": "Missing user_prompt"}), 400

    ide = payload.get("ide", {})
    files = ide.get("files", {})
    if not isinstance(files, dict) or not files:
        return jsonify({"success": False, "error": "Missing ide.files
(multi-file context required)"}), 400

    system_prompt = build_ai_system_prompt(ide)

    try:
        completion = client.chat.completions.create(
            model=OPENAI_MODEL,
            messages=[
                {"role": "system", "content": system_prompt},
                {"role": "user", "content": user_prompt},
            ],
            temperature=0.15,
            max_tokens=1100,
        )

        raw = completion.choices[0].message.content or ""
        ok, obj, err = safe_json_loads(raw)
```

```
        if not ok:
            return jsonify({
                "success": False,
                "error": f"AI returned invalid JSON: {err}",
                "raw": raw[:2000]
            }), 200

        # Validate structure
        patches = obj.get("patches", [])
        message = obj.get("message", "")
        suggested_actions = obj.get("suggested_actions", {}) or {}
        safety = obj.get("safety", {}) or {}

        if not isinstance(patches, list):
            return jsonify({"success": False, "error": "AI JSON missing 'patches' array", "raw": obj}), 200

        # Basic size guard
        patch_chars = sum(len((p.get("diff") or "")) for p in patches if isinstance(p, dict))
        if patch_chars > MAX_PATCH_CHARS:
            return jsonify({
                "success": False,
                "error": f"Patches too large ({patch_chars} chars). Refuse.",
                "raw": obj
            }), 200
```

```
        # DRY-RUN validate: apply patches server-side, scan for forbidden +
physics risks

        ok_apply, new_files, warnings, errors =
validate_and_simulate_patchset(files, patches)



        blocked = not ok_apply

        if blocked:

            return jsonify({

                "success": True,

                "message": message or "AI produced changes but they were blocked
by safety validation.",

                "patches": [],  # do not send unsafe patches

                "suggested_actions": {"run_after_apply": False, "focus_file":
ide.get("current_file", "main.js")},

                "safety": {

                    "warnings": warnings,

                    "blocked": True,

                    "errors": errors

                }

            }), 200



        # Not blocked. Return patches to client to apply.

        return jsonify({

            "success": True,

            "message": message or "Patches ready.",

            "patches": patches,

            "suggested_actions": {
```

```
                "run_after_apply": bool(suggested_actions.get("run_after_apply",
False)),

                "focus_file": str(suggested_actions.get("focus_file") or
ide.get("current_file") or "main.js")

            },

            "safety": {

                "warnings": warnings,

                "blocked": False

            }

        }), 200



    except Exception as e:

        return jsonify({"success": False, "error": str(e)}), 500




@app.route("/health")

def health():

    return jsonify({"status": "ok", "ai": True, "model": OPENAI_MODEL})




# Optional static passthroughs

@app.route("/physics/<path:filename>")

def serve_physics(filename):

    base = os.path.join(os.getcwd(), "physics")

    if not os.path.exists(os.path.join(base, filename)):

        abort(404)

    return send_from_directory(base, filename)
```

```python
if __name__ == "__main__":
    port = int(os.getenv("PORT", 5000))
    host = os.getenv("HOST", "127.0.0.1")
    print(f"Starting server on http://{host}:{port}")
    app.run(host=host, port=port, debug=True)
```